R-IN32M3 Module (RY9012A0)

Modbus TCP スタートアップマニュアル

R30AN0406JJ0200
Rev.2.00
2022.08.05

要旨

R-IN32M3 Module (RY9012A0)を使った Modbus TCP 通信を行うためのスタートアップマニュアルです。

本資料では、R-IN32M3 モジュール搭載 RX66T CPU カード[SEMB1320] 評価環境を例に説明します。

動作確認デバイス

R-IN32M3 Module (RY9012A0)

目次

1. 概要 3
1.1 特長 3
1.2 動作環境 4
1.2.1 ハードウェア環境 4
1.2.2 ソフトウェア環境 5
1.3 参考ドキュメント 6

2. サンプルアプリケーション 7
2.1 システム構成 7
2.1.1 uGOAL 7
2.1.2 Modbus スタックプログラム 8
2.1.3 Modbus アプリケーション 8
2.2 ブロック図 9
2.3 ファイル構成 10
2.4 ビルド構成 10
2.5 リソース構成 11
2.6 Modbus スタック概要 12
2.6.1 Modbus TCP サーバスタック 12
2.6.2 Modbus TCP ゲートウェイ 13
2.6.3 パケット解析 14
2.6.4 TCP/IP 通信管理 16

3. アプリケーションインプリガイド 17
3.1.1 データマッピング 17
3.1.2 uGOAL 構成部 20
3.1.3 Modbus スタック構成部 26
3.1.3.1 コンフィグ設定 26
3.1.3.2 スタック初期化 28
3.1.3.3 ファンクションコード対応 29
3.1.3.4 TCP/IP 通信管理 32
3.1.4 シリアル通信部 33

4. 評価用ツールを用いた通信テスト 34
4.1 Modbus TCP サーバ接続 34
4.2 Modbus TCP ゲートウェイ接続 38

Appendix 41
A. ファンクションコード用構造体 41
B. API 仕様書 46
C. サンプルソフトウェア起動方法 57

1. 概要

本書は、R-IN32M3 Module と接続したホストマイコンで動作する Modbus プロトコルスタックの資料であり、プロトコルスタックを使ったアプリケーションを開発、実装する際の機能概要やアプリケーション・プログラミング・インタフェース（API）、アプリケーションサンプルについて記載しています。

本パッケージでは、イーサネット・ベースの Modbus TCP スレーブに対応しています。

1.1 特長

Modbus プロトコルは、Modicon Inc.（Schneider Electric SA.）がプログラマブルロジックコントローラ（PLC）向けに開発した通信プロトコルであり、仕様は公開されています。

プロトコル仕様書（PI-MBUS-300 Rev.J）を参照ください。

R-IN32M3 Module 向けホストマイコン用 Modbus プロトコルスタックは、次のアプリケーションの迅速かつ容易な開発を可能とします。

- Modbus TCP サーバ
- Modbus TCP ゲートウェイ

R-IN32M3 Module 向けホストマイコン用 Modbus プロトコルスタックでは、以下の 9 つのファンクションコードをサポートします。

- 1 (0x01) – Read coils
- 2 (0x02) – Read discrete input
- 3 (0x03) – Read holding registers
- 4 (0x04) – Read input registers
- 5 (0x05) – Write single coil
- 6 (0x06) – Write single register
- 15 (0x0F) – Write multiple coils
- 16 (0x10) – Write multiple registers
- 23 (0x17) – Read/Write multiple registers

Modbus に関する詳細は、以下のサイトを参照して下さい。

http://www.modbus.org

「Modicon Modbus Protocol Reference Guide Rev.J」 (PI_MBUS_300.pdf)

「Modbus Application Protocol Specification V1.1b3」(Modbus_Application_Protocol_V1_1b3.pdf)

※更新によりバージョン番号は異なる場合がございます。最新のマニュアルを参照ください。

1.2 動作環境

動作環境について説明いたします。

1.2.1 ハードウェア環境

R-IN32M3 モジュールと接続して使用するホストマイコンを使った Modbus TCP 制御は下記のいずれかの構成を対象としています。

本資料では、R-IN32M3 モジュール搭載 RX66T CPU カード[SEMB1320] 評価環境を例に説明します。

(1) R-IN32M3 モジュール搭載 CPU カード [SEMB1320]

(2) R-IN32M3 モジュール搭載アダプタボードと EK-RA6M3 または EK-RA6M4 との組合せ

(3) R-IN32M3 モジュール搭載アダプタボードと RL78/G14 Fast Prototyping Board との組合せ

(1) R-IN32M3 Module RX66T CPU Card [SEMB1320]

(2) Adapter Board + EK-RA6M4

(3) Adapter Board + RL78/G14 Fast Prototyping Board

図 1-1 R-IN32M3 モジュール 開発環境

表 1-1 ハードウェア環境

Name	Type Name	Maker	Link
R-IN32M3 Module RX66T CPU Card	SEMB1320	SHIMAFUJI Electric Incorporated	R-IN32M3_Module-SEMB1320
Adapter Board	YCONNECT-IT-I-RJ4501	Renesas Electronics Corporation	R-IN32M3-Module-Solution-Kit
EK-RA6M3	RTK7EKA6M3S00001BU	Renesas Electronics Corporation	RA6M3 MCU Group Evaluation Board
EK-RA6M4	RTK7EKA6M4S00001BE	Renesas Electronics Corporation	RA6M4 MCU Group Evaluation Board
RL78/G14 Fast Prototyping Board	RTK5RLG140C00000BJ	Renesas Electronics Corporation	RL78/G14 fast Prototyping Boad

1.2.2 ソフトウェア環境

表 1-2 ソフトウェア環境

Category	Name	Version	Link	備考
R-IN32M3 モジュール サンプルパッケージ	サンプルパッケージ	Rev.1.**	https://www.renesas.com/	
RA 環境				
統合開発環境	e2studio	2022-04	e² studio 2022-04 Windows| Renesas	
Flexible Software Package	FSP	V3.6.0	-	e2studio インストーラーに同梱 FSPv3.6.0 | GitHub
GNU Arm Embedded Toolchain	GCC Toolchain	V10.3.1.20210824	—	e2studio インストーラーに同梱
RX 環境				
統合開発環境	e2studio	2022-04	e² studio 2022-04 Windows| Renesas	
RX ファミリ用 GNU Toolchain	GCC for Renesas RX	V8.3.0.202102	—	e2studio インストーラーに同梱
RL 環境				
統合開発環境	e2studio	2022-04	e² studio 2022-04 Windows| Renesas	
RL ファミリ用 GNU Toolchain	GCC for Renesas RL78	V4.9.2.202103	—	e2studio インストーラーに同梱

1.3 参考ドキュメント

Modbus に関する技術情報は Modbus Organization のサイトから、R-IN32M3 Module に関する情報はルネサス エレクトロニクスのサイトから入手できます。

・Modbus Organization のサイト : http://www.modbus.org

・ルネサス エレクトロニクスのサイト : http://www.renesas.com

表 1-3 Modbus 関連ドキュメント

項番	ドキュメント
1	Modbus_Application_Protocol_V1_1b3.pdf
2	PI_MBUS_300.pdf
3	Modbus_over_serial_line_V1_02.pdf
4	Modbus_Messaging_Implementation_Guide_V1_0b.pdf

表 1-4 R-IN32M3 Module 関連ドキュメント

資料名	資料番号
R-IN32M3 Module (RY9012A0) データシート	R19DS0109JD****
R-IN32M3 Module (RY9012A0) ユーザーズマニュアル ハードウェア編	R19UH0122JD****
R-IN32M3 Module (RY9012A0) ユーザーズマニュアル ソフトウェア編	R17US0002JD****
R-IN32M3 Module (RY9012A0) ユーザ実装ガイド (uGOAL編)	R30AN0402JJ****
R-IN32M3 Module (RY9012A0) 搭載アダプタボード YCONNECT-IT-I-RJ4501 ユーザーズマニュアル	R12UZ0094JJ****
Management Tool 操作ガイド	R30AN0390JJ****
RA サンプルアプリケーション (uGOAL版)	R30AN0398JJ****
RX66T サンプルアプリケーション(uGOAL版)	R30AN0399JJ****
RL78/G14サンプルアプリケーション (uGOAL版)	R30AN0400JJ****

2. サンプルアプリケーション

2.1 システム構成

本サンプルアプリケーションは、大きく分けて 3 つのブロックに分かれます。

1. 産業通信ドメインでのアプリケーション構築に利用可能な uGOAL と呼ばれるソフトウェアスタック
2. uGOAL の TCP/IP プロトコルスタックを使用する Modbus プロトコルスタックサンプルプログラム
3. uGOAL および Modbus プロトコルスタックを使用するアプリケーションサンプルプログラム

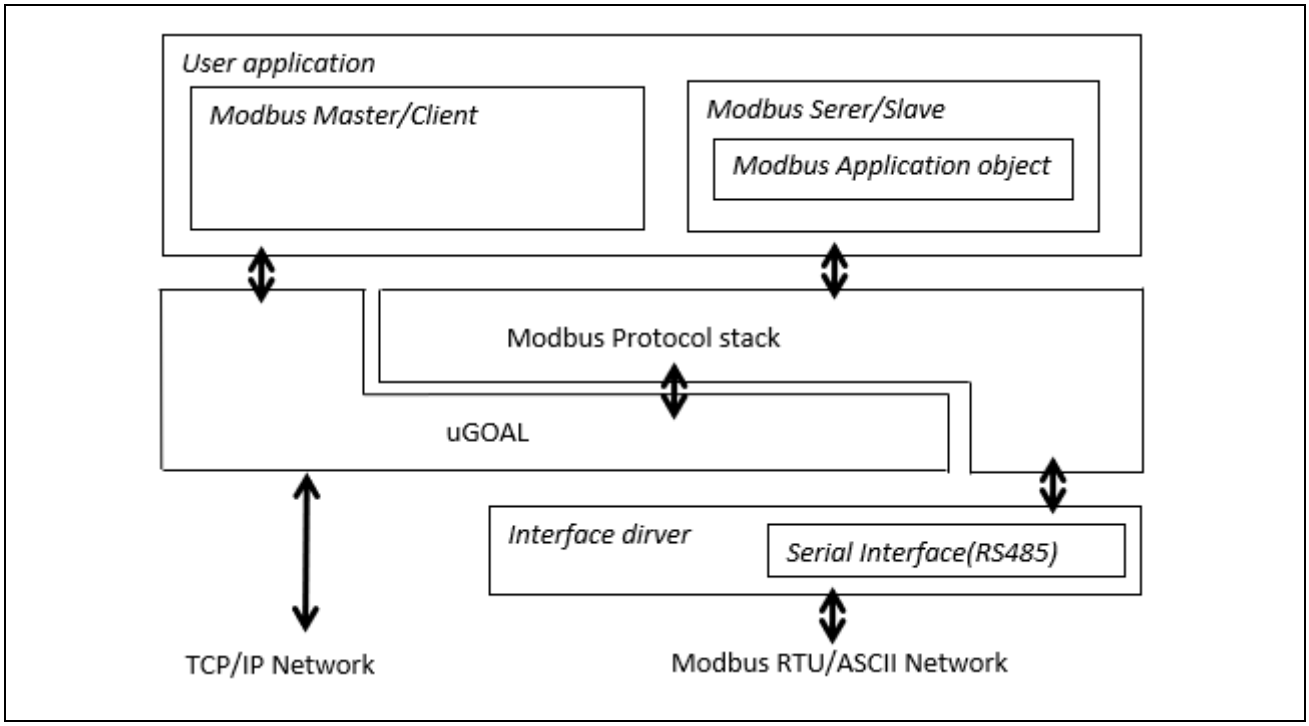


図 2-1 システム構成

2.1.1 uGOAL

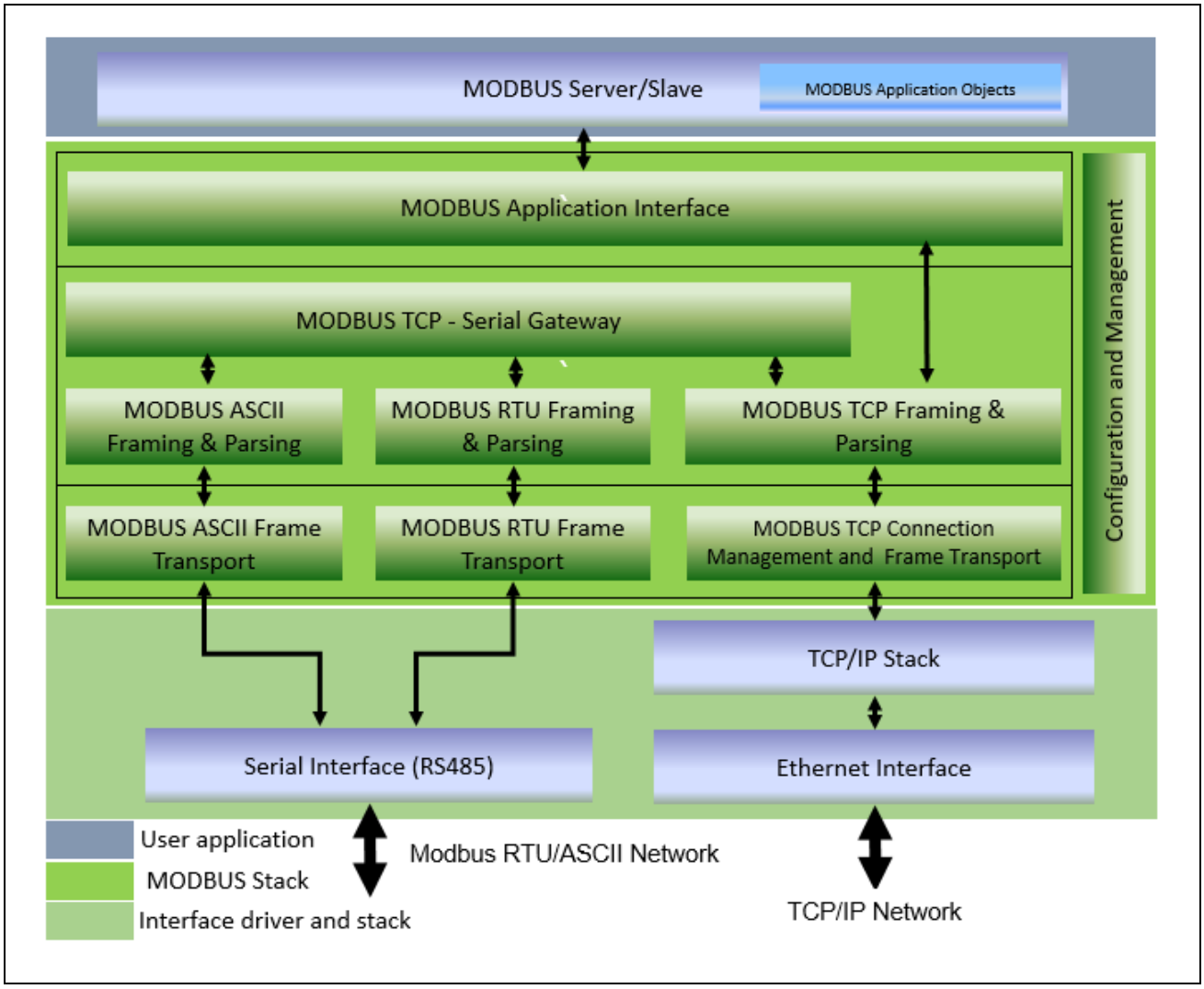
R-IN32M3 Module のソフトウェアには、産業通信ドメインでのアプリケーション構築に利用可能なソフトウェアスタックで構成される、uGOAL（ドメイン固有のミドルウェア）が含まれています。本サンプルプログラムでは、TCP/IP スタック制御として使用します。

uGOAL の詳細については、『R-IN32M3 Module (RY9012A0) ユーザーズマニュアル ソフトウェア編 (R17US0002JJ****)』を参照ください。

2.1.2 Modbus スタックプログラム

本サンプルアプリケーションは、Modbus プロトコルに基づく通信機能を提供するプロトコルスタックのサンプルプログラムを含みます。

本サンプルプログラムは、TCP/IP 通信として uGOAL を使用します。また、Modbus TCP シリアルゲートウェイモードでは、シリアルネットワークへのゲートウェイとして Modbus RTU/ASCII マスタスタックとして振舞います。Modbus RTU/ASCII マスタスタックの初期化は、ゲートウェイスタックの初期化内で行われます。ユーザは、Modbus RTU または Modbus ASCII ゲートウェイスタックのいずれかを選択することができます (シリアルモード選択：MBAPP_INIT_GW_SERIAL_MODE)。



2.1.3 Modbus アプリケーション

本サンプルアプリケーションは、uGOAL および Modbus プロトコルスタックのサンプルプログラムを用いて、Modbus プロトコル通信をデモンストレーションするサンプルプログラムです。

詳細は、2.6 章「Modbus スタック概要」を参照ください。

2.2 ブロック図

RX66T 向けサンプルアプリケーションのブロック図を示します。

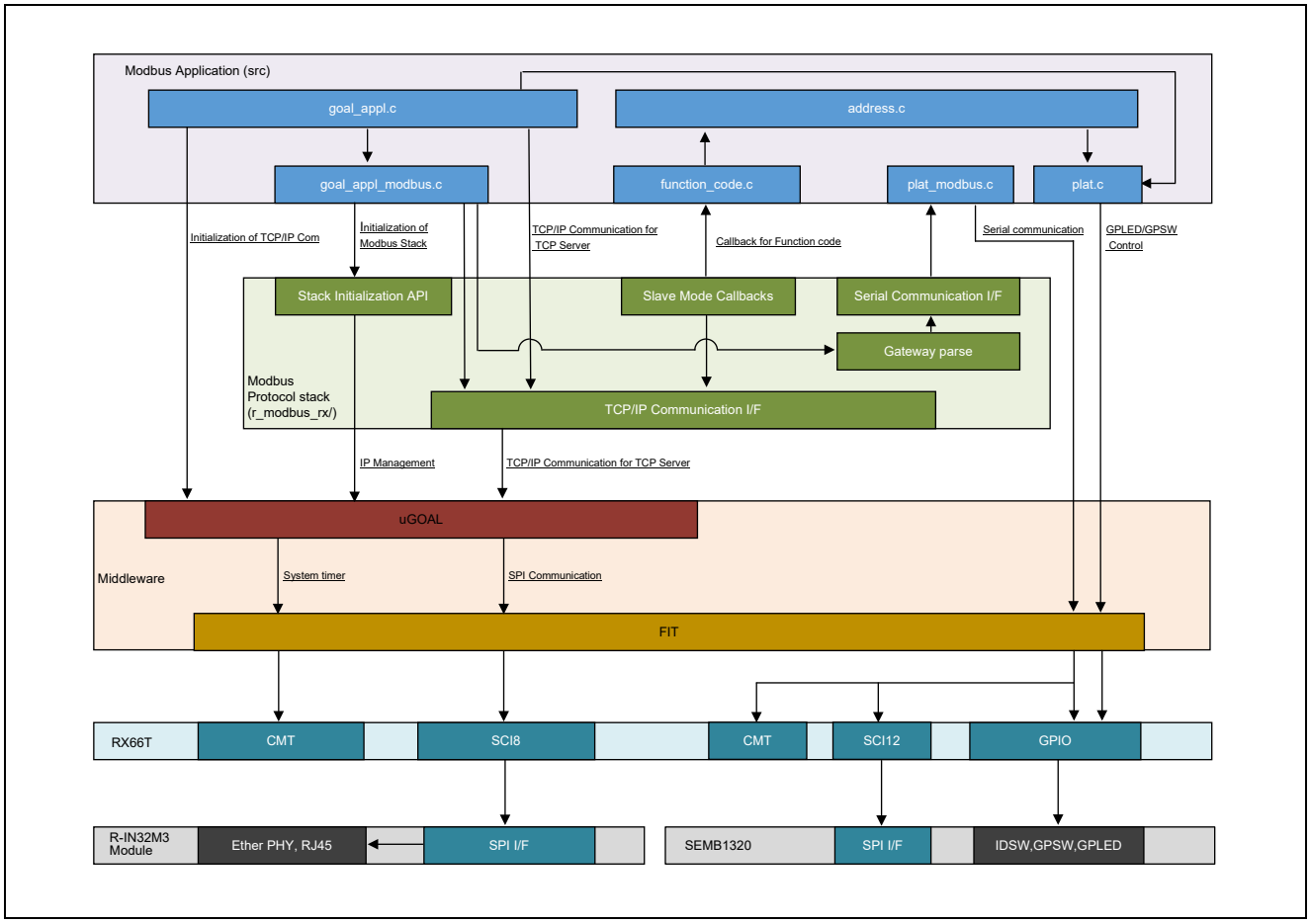


図 2-2 RX66T サンプルアプリケーション機能ブロック図

2.3 ファイル構成

ファイル構成（一部省略）を表 2-1 に示します。

表 2-1 ファイル構成

フォルダ/ファイル名	説明
07_mbus_tcp_server/	Modbus アプリケーション サンプルプログラム
goal_appl.c	uGOAL アプリケーション部実装
goal_config.h	uGOAL コンフィグ設定ヘッダファイル
goal_appl_modbus.c (.h)	Modbus プロトコルスタック 初期化およびパケット解析部実装
mbapp_slave/	Modbus スレーブモード用実装
function_code.c (.h)	Modbus スレーブモード用ファンクションコードコールバック実装
address.c (.h)	Modbus スレーブモード用データアドレス実装
r_modbus/	Modbus プロトコルスタック サンプルプログラム
src/ (inc/)	Modbus プロトコルスタック サンプルプログラム ソース＆ヘッダファイル

2.4 ビルド構成

本サンプルアプリケーションでは、Modbus プロトコルの各動作モードに対応したビルド構成を準備しています。

プログラム実行させるビルド構成の選択は、Appendix. C 手順 4) を参照ください。

表 2-2 ビルド構成一覧

スタックモード	ビルド構成名
Modbus TCP サーバスタック	TCP_SERVER_UGOAL
Modbus TCP ゲートウェイ機能付きサーバスタック	TCP_GATEWAY_UGOAL

2.5 リソース構成

本サンプルプログラムで使用するハードウェアリソースについて説明します。

(1) 使用するモジュール

Modbus スタックとして使用する RX66T 搭載 SEMB1320 評価ボードのハードウェアリソースを表 2-3 に示します。本サンプルプログラムに機能を追加する場合は、リソースの競合にご注意ください。

表 2-3 H/W モジュール一覧

SW ブロック	HW モジュール	入出力端子	用途
Modbus プロトコルスタック	SCI12	P21（TXD12） P22（RXD12）	Modbus RTU/ASCII シリアル（RS485）通信
	汎用 I/O ポート	P20	Modbus RTU/ASCII シリアル通信 RS485 トランシーバ方向制御用端子
	CMT	-	RTU/ASCII モード RS485 通信タイミング制御
Modbus アプリケーション	汎用 I/O ポート	PE3 PE4 PE1 PE0	LED 制御 （SEMB1320 評価ボード LED5,LED6,LED8,LED9）
		PB4 PB2 PB1 PB0	スイッチ入力 （SEMB1320 評価ボード SW2,SW4,SW5,SW6）

* CMT: Compare match timer , SCI: Serial communication interface

(2) ヒープサイズの設定

uGOAL 管理下のヒープ領域を使用します。必要に応じて、plat フォルダ配下の plat.h に含まれる ”CONFIG_UGOAL_HEAP_BUFFER_SIZE” のマクロ値を変更します。

2.6 Modbus スタック概要

本サンプルソフトは、uGOAL ミドルウェアを備えており、その設計思想に基づいた構成となっています。さらに、本サンプルソフトでは、uGOAL で発生した TCP 情報を Modbus スタック部に受け渡す構造になっています。

2.6.1 Modbus TCP サーバ スタック

Modbus TCP サーバスタックモード時 [ビルド構成：TCP_SERVER_UGOAL] におけるプログラム全体の流れを示します。

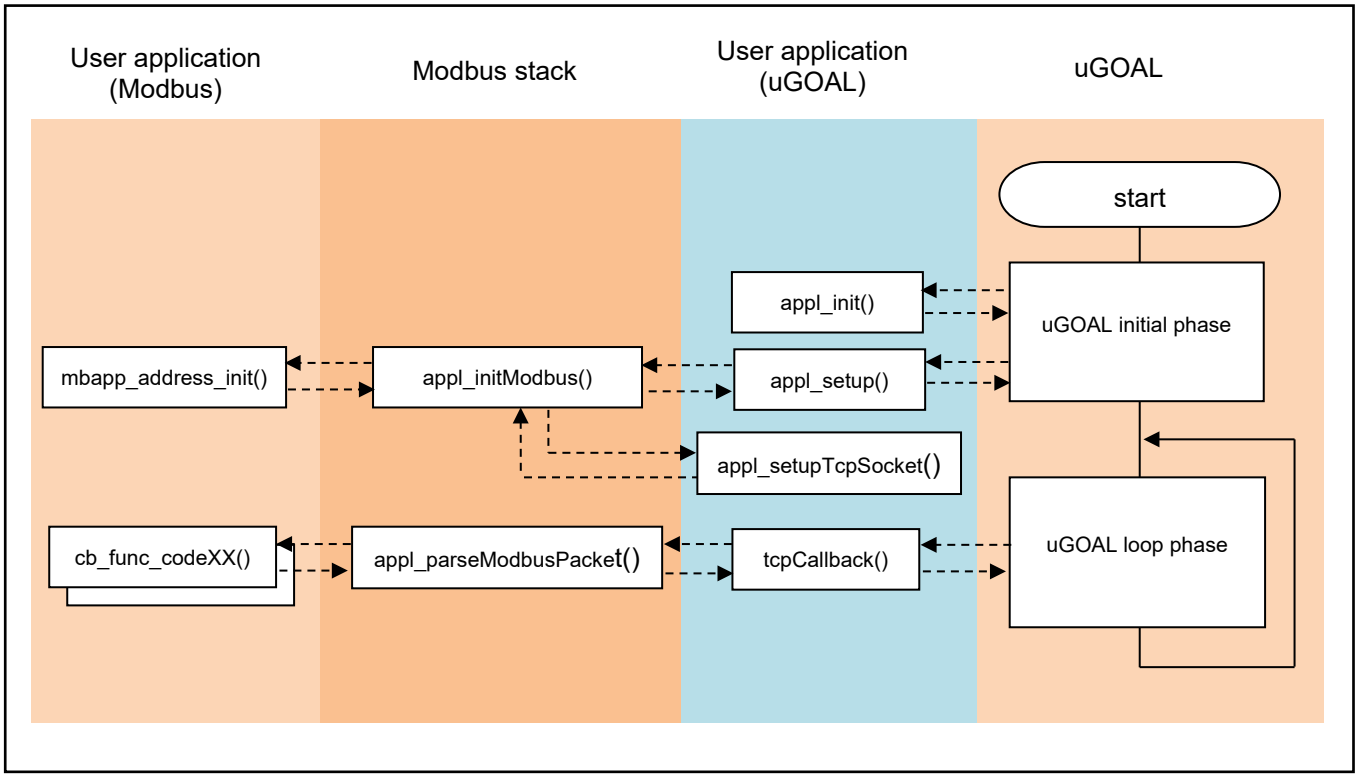


図 2-3 プログラム全体の流れ Modbus TCP サーバ

uGOAL ではユーザアプリケーション固有の処理のために、*appl_init()*、*appl_setup()* の関数が用意されています。また、本サンプルでは TCP プロトコルのイベント処理用として *tcpCallback()* をコールバック関数として登録しています。uGOAL の初期フェーズで *appl_init()* と *appl_setup()* が実行され、その後のループフェーズで TCP の通信イベントが発生する度に *tcpCallback()* が実行される構成となっています。パケット処理詳細については、2.6.3 章を参照してください。

uGOAL にて発生した TCP イベントを契機に、Modbus スタック部にデータを受け渡し、Modbus TCP としての処理を行います。Modbus スタック部では、スタックの初期設定を行う *appl_initModbus ()* と、パケット解析を行う *appl_parseModbusPacket()* の 2 つに大まかに分けられます。本サンプルでは、ユーザ実装部分として、mbapp_address_init() および cb_func_code01()等が用意されています。ユーザは、各デバイスや実装したい機能に応じて、各関数の実装を行います。処理詳細については、3.1.2 章を参照してください。

2.6.2 Modbus TCP ゲートウェイ スタック

Modbus ゲートウェイ モード時 [ビルド構成：**TCP_GATEWAY_UGOAL**] におけるプログラム全体の流れを示します。

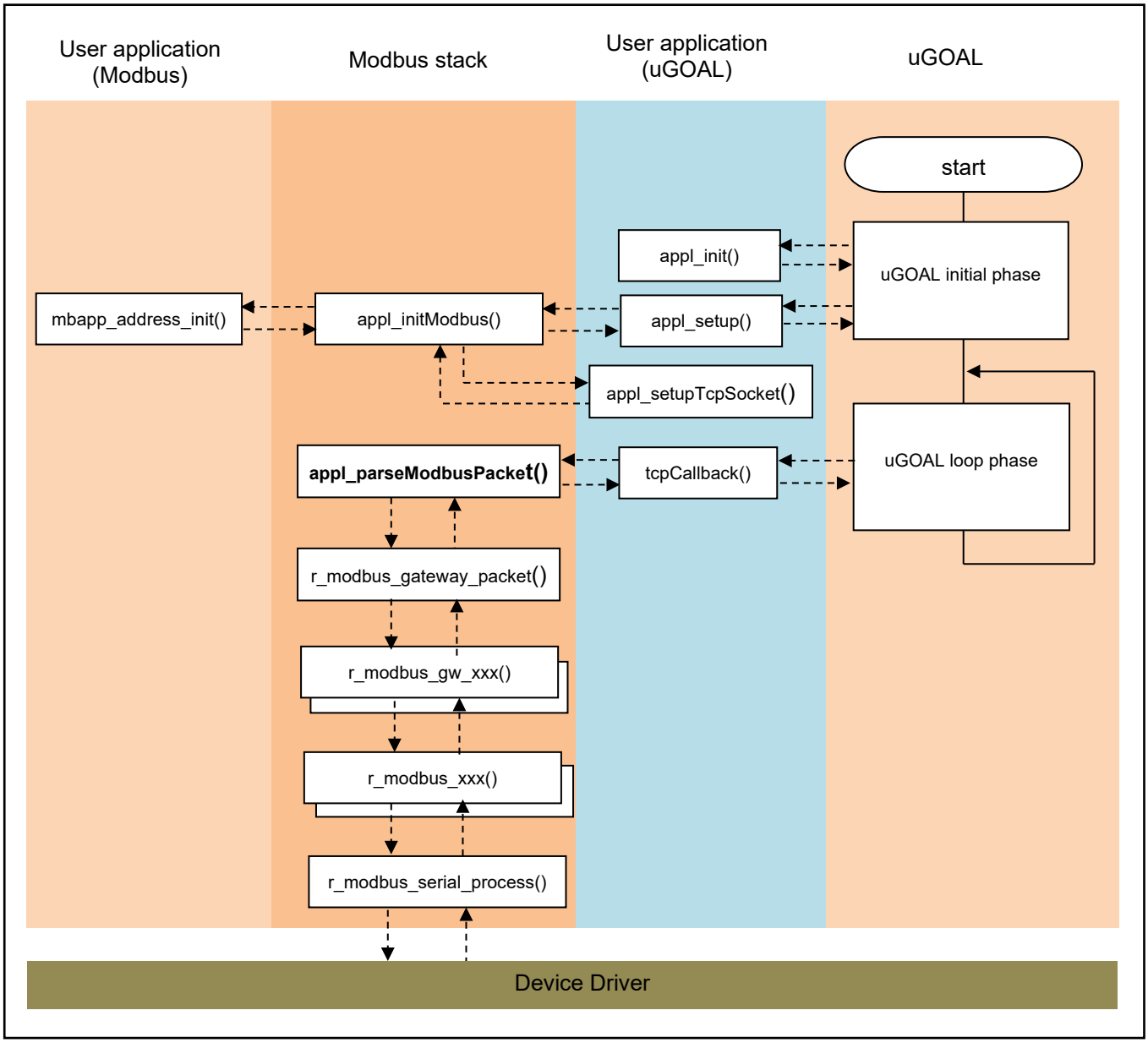


図 2-4 プログラム全体の流れ Modbus TCP ゲートウェイ

Modbus ゲートウェイ モード時における基本的な流れは、Modbus TCP と変わりませんが、Modbus パケット解析後の動作が変化します。***appl_parseModbusPacket*** でゲートウェイパケットと認識すると、ゲートウェイパケット処理に移行します。

ゲートウェイパケット処理では、クライアントから受信したリクエストパケットを、RTU または ASCII モードのリクエストパケットに変換し、接続されているシリアルデバイスに対して送信します。送信後、リクエストに対する応答を待ち、受信したレスポンスパケットを今度は、Modbus TCP のレスポンスパケットに変換して、クライアントに送信します。

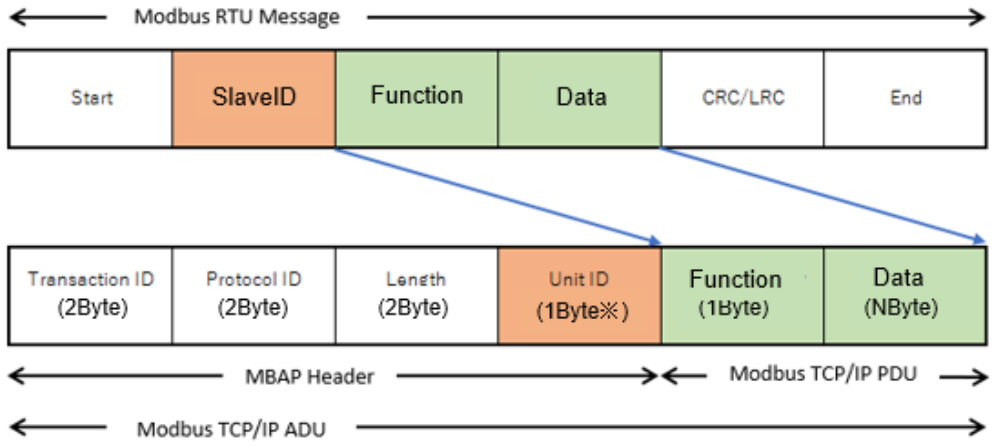
2.6.3 パケット解析

uGOAL で受信した TCP パケットを解析し、各ファンクションコードに実装された機能を実行します。尚、本節で説明している機能は、サンプルソフト上の関数 *appl_parseModbusPacket* にて実装されています。

Modbus TCP の通信フォーマットは、Modbus RTU の CRC を除く部分を含んだ形となっています。Modbus RTU では誤りチェックのため、CRC チェックコードを末尾に付加する必要がありましたが、Modbus TCP では TCP/IP プロトコルが持つチェック機構を利用するため不要となります。

Modbus TCP の通信フォーマットでは、Modbus RTU には存在しなかった、トランザクション識別子、プロトコル識別子、メッセージ長、ユニット識別子を付加する必要があります。

Modbus RTU フォーマットと Modbus TCP/IP フォーマットの違いを示します。



ADU：Application Data Unit **MBAP**：Modbus Application **PDU**：Protocol Data Unit

図 3-5 Modbus RTU フォーマットと Modbus TCP/IP フォーマット

前述の通信フォーマットに従い、受信した TCP パケットの解析を行います。以下にパケット解析時の要点を示します。

- 受信パケットの検証で、長さチェック、パケットの整合性およびスレーブ ID の不一致などが発生した場合、受信パケットを破棄します。
- 受信パケットが正常ならば、リクエストを処理するために、ユーザが登録したコールバック関数を呼び出します。
- ユニキャスト要求を受信した場合、ファンクションコードの処理に異常があれば、例外応答メッセージを作成しクライアントに送ります。
- ブロードキャスト要求を受信した場合、受信したパケットがライト系ファンクションコードならば正常パケットとして受け付けますが応答メッセージをクライアントに送信しません。また、受信したパケットがリード系ファンクションコードならば、スレーブ ID 異常として要求パケットを破棄します。

ゲートウェイモードの場合、受信した Modbus TCP パケットを解析して、自局（スレーブ ID が一致）への指示でなければ、接続されたシリアルデバイスへの指示とみなし、対象デバイスへシリアル通信を行います。

・受信した Modbus TCP パケットから Modbus RTU パケットを生成します。
・シリアルモードが ASCII モードの場合、RTU→ASCII 形式へのパケット変換を行います。
・構築したパケットに、CRC/LRC を付加してパケットを送信します。
・送信後、シリアルモードに応じて応答パケットの受信を待ちます。
・シリアルモードが ASCII モードの場合、ASCII→RTU 形式へのパケット変換を行います。
・応答パケットの CRC の検証を行い、パケットの長さ等のチェックを行います。
・エラーがなければ、Modbus RTU パケットから Modbus TCP パケットを生成して、送信します。

2.6.4 TCP/IP 通信管理

本サンプルプログラムは、Modbus スタック上で接続される TCP クライアントを管理しています。本節では、TCP/IP 接続管理に関する機能について説明します。

(1) IP アドレスリスト機能

当該 TCP サーバに接続するクライアントを IP アドレス単位で許可（ホワイトリスト）および拒否（ブラックリスト）する機能をサポートします。
リスト機能を有効とした場合[**MBAPP_INIT_IP_TABLE_FLAG**]、動作モードに従い、接続要求のあったクライアントの IP アドレスをチェックして、接続要求を受け付けるか拒否するかを決定します。サンプルソフト上では関数 *appl_tcpCallback* で実装されています。
なお、対象とする IP アドレスは、ユーザ側で予め登録しておく必要があります。登録可能な IP アドレス数は、**MAXIMUM_NUMBER_OF_CLIENTS** に依存します。

(2) マルチクライアント機能

マルチクライアント機能を有効とした場合[**ENABLE_MULTIPLE_CLIENT_CONNECTION**]、FIFO 方式で **MAXIMUM_NUMBER_OF_CLIENTS** で指定されたクライアント数まで接続を受け付けます。最大値を超えて接続要求があった場合は、最も古い接続を切断したのち、新しいクライアントからの接続を受け付けます。マルチクライアント機能を無効とした場合、２つ目以降のクライアントからの接続要求はすべて拒否します。
Modbus スタックでは、同時接続可能な TCP クライアントの数を管理しています。
TCP の接続・切断を検出した場合、Modbus スタック側にも報告します。Modbus スタック側で、接続上限数に達している場合、最も古い接続を切断したのち、新たな接続を有効な接続として登録します。
尚、本節で説明する内容は、サンプルソフト上では関数 *tcpCallback* で実装されています。

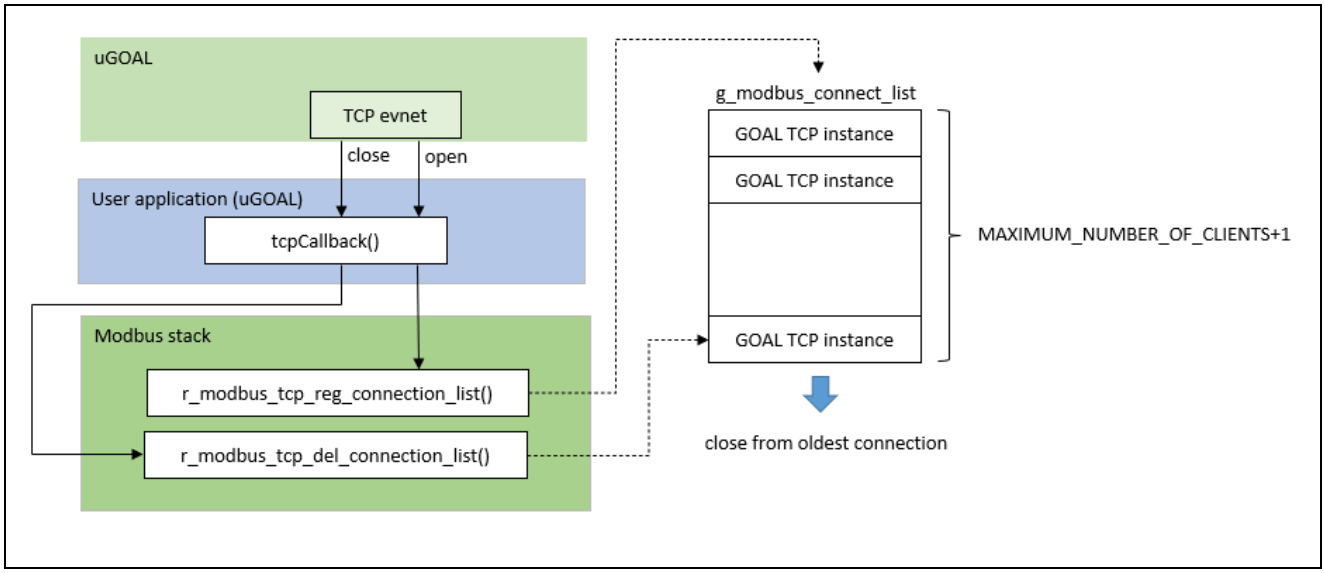


図 3-6 TCP 接続管理

3. アプリケーションインプリガイド

ユーザアプリケーションとして固有の処理を実装する際の手順を、サンプルプログラムを例に説明します。

3.1 データマッピング

本サンプルで実装している Modbus データモデルについて説明します。尚、本節で説明する内容は、各サンプルアプリケーションの address.c(.h)にて定義されています。

- 各 Modbus データ型は、2048 個のデータを持ち、参照アドレス 0x0001~0x0800 を使用します（※1）。
- 参照アドレスは、静的グローバル変数の配列バッファとして RAM メモリに割り当てます。
- Modbus ファンクションコードのコールバック関数で使用される、各 Modbus データ型の各配列バッファにアクセス（Read/Write）する関数を実装しています。
- Read/Write アクセス関数は、特定の参照アドレスにアクセスした場合、バッファ配列と各デバイスの周辺機能レジスタへリンクします（※2）。
  - Coils アドレス 0x0001~0x0004 番地 ： LED に対応する GPIO ポート
  - Discrete Input アドレス 0x0001~0x0004 番地：SW に対応する GPIO ポート
- 各 Modbus データアドレスの 0x07D1 番地にアクセスした場合に、回復不能なエラーが発生したとして、例外コード 0x04：SLAVE DEVICE FAILURE を返します（※3）。
- 各 Modbus データ型で指定したアドレス範囲に、不正なアドレスが含まれているかをチェックする関数を実装しています。
- 各 Modbus ファンクションコードに対応するコールバック関数は、これらのチェック関数を使用することで、バッファ配列にアクセスする前に、不正なアドレスを検知して例外コード 0x02：ILLEGAL DATA ADDRESS を返すことができます（※4）。

各々の Modbus データ型の参照アドレス設計、および物理メモリマッピングを図 2-6 に示します。

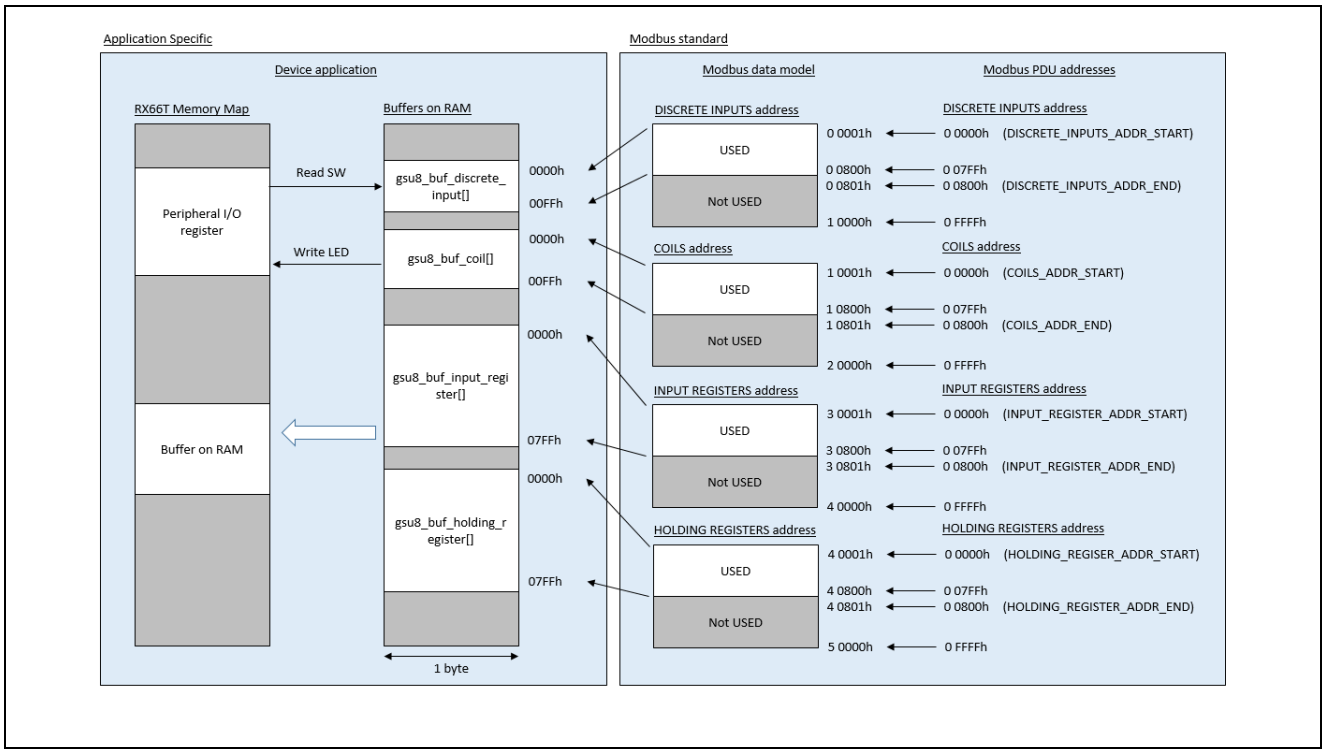


図 3-1 参照アドレスおよびマッピング

※1 ：使用する参照アドレスの範囲は、各 Modbus データ型に対応する定数マクロで指定できます。

- 例えば、COILS_ADDR_START、COILS_ADDR_END 定数マクロは、データ型 Coils の参照アドレスの開始アドレスと終了アドレスを指定できます。
- デフォルトでは、各 Modbus データアドレスに開始アドレスとして 0x0000、終了アドレスとして 0x0800 を指定しています。
- ここで指定されるアドレス値は、0x0001 開始ではなく、0x0000 開始であることに注意します。
- これは、Modbus リクエストを構成するプロトコルデータユニット（PDU）に記載されているアドレス値表現に準拠するためです。以下、これアドレス表現を PDU アドレスと呼びます。
- このサンプルプログラム上では、参照アドレスを全て PDU アドレスで処理をしています。
- また、COILS_ADDR_END などの有効アドレス終了指定値は排他としており、指定した値のアドレスは無効なアドレスとなることに注意します。

※2 ：本サンプルプログラムでは、各デバイスの周辺機能レジスタへ割り当てられる参照アドレスは、それぞれ COILS_ADDR_START、および DISCRETE_INPUT_ADDR_START との相対値で決定しています。

- 例えば、COILS_ADDR_START を 0x1000 にした場合、Coil アドレス 0x1001~0x1004 番地が、SEMB1320 ボード上の LED5,6,8,9 に対応する GPIO ポートとリンクします。

※3： 例外コード 0x04 を返すアドレス値は、定数マクロ PUSEDO_SLAVE_FAILURE_ADDR で指定できます。指定値は PDU アドレスであることに注意します。

※4：不正アドレスのチェック関数を変更することで、任意のアドレスを不正なアドレスとして設計することができます。

以下に Read Coil（ファンクションコード 01h）実行時にコールバック関数から呼び出される Coils 領域へのデータアクセス関数のサンプルコード例を示します。

例) ファイル: *07_mbus_tcp_server\mbapp_slave\address.c*

```
uint8_t read_coil_address(uint16_t u16_address, uint8_t *u8_bit)
{
        if( u16_address == PUSEDO_SLAVE_FAILURE_ADDR )
        {
                return ERR_SLAVE_DEVICE_FAILURE;
        }

        /* calculate buffer index from address structure */
        uint16_t buf_addr = u16_address - COILS_ADDR_START;

        /* assign LEDs to coil address ADDR_START ~ ADDR_START+3  */
        if( buf_addr < GPLED_NUM )
        {
                /* write LEDs data in buffer */
                _write_bit_buf(gsu8_buf_coil, buf_addr, plat_remoteIoLedGet(buf_addr));
        }

        /* read buffer */
        *u8_bit = _read_bit_buf(gsu8_buf_coil, buf_addr);

        return ERR_OK;
}
```

①

②

③

①特定アドレスの場合にデバイスエラーを返すテストコードになります。

②Coils 領域に対応したデバイス（この場合 Remote I/O 用 LED）の状態を読み取ります。

③Coils 領域のデータをバッファに格納します。

3.1.1 uGOAL 構成部

サンプルアプリケーションにおける uGOAL ミドルウェアに関連する部分の実装について説明します。本節で説明する内容は、各サンプルアプリケーションの goal_appl.c にて定義されています。尚、各 API の詳細については、『R-IN32M3 Module (RY9012A0) ユーザーズマニュアル ソフトウェア編 (R17US0002JJ****)』を参照ください。

(1) appl_init

本関数には、uGOAL コアモジュール等が初期化される前に、アプリケーション固有の初期化ステップを含めます。Modbus サンプルでは、TCP/IP を使用して通信を行うため、ネットワーク機能の初期化を行っています。

```
GOAL_STATUS_T appl_init(
    void
)
{
    GOAL_STATUS_T res;                          /* result */

    /* initialize goal net */
    res = goal_netRpcInit(GOAL_NET_ID_DEFAULT);
    if (GOAL_RES_ERR(res)) {                                                    ①
        goal_logErr("Initialization of goal net RPC failed");
    }

    return res;
}
```

①ネットワークの RPC 機能の初期化を行います。

(2) appl_setup

本関数では、ネットワーク設定および Modbus スタックの初期化を行います。

```
GOAL_STATUS_T appl_setup(
    void
)
{
    GOAL_STATUS_T res;                          /* result */
    uint32_t ip;                                /* IP address */
    uint32_t nm;                                /* netmask */
    uint32_t gw;                                /* gateway */
    uint32_t cnt;                               /* counter */

    goal_maLedSet(pMaLed, GOAL_MA_LED_LED1_RED, GOAL_MA_LED_STATE_OFF);
    goal_maLedSet(pMaLed, GOAL_MA_LED_LED1_GREEN, GOAL_MA_LED_STATE_OFF);
    goal_maLedSet(pMaLed, GOAL_MA_LED_LED2_RED, GOAL_MA_LED_STATE_OFF);
    goal_maLedSet(pMaLed, GOAL_MA_LED_LED2_GREEN, GOAL_MA_LED_STATE_OFF);
    goal_maLedSet(pMaLed, GOAL_MA_LED_ETHERCAT, GOAL_MA_LED_STATE_OFF);
    goal_maLedSet(pMaLed, GOAL_MA_LED_PROFINET, GOAL_MA_LED_STATE_OFF);
    goal_maLedSet(pMaLed, GOAL_MA_LED_MODBUS, GOAL_MA_LED_STATE_ON);
    goal_maLedSet(pMaLed, GOAL_MA_LED_ETHERNETIP, GOAL_MA_LED_STATE_OFF);                    ①

    res = goal_maNetOpen(GOAL_NET_ID_DEFAULT, &pMaNet);
    if (GOAL_RES_ERR(res)) {
        goal_logErr("error opening network MA");                                             ②
    }

    /* set IP address */
    ip = MAIN_APPL_IP;
    nm = MAIN_APPL_NM;
    gw = MAIN_APPL_GW;
    res = goal_maNetIpSet(pMaNet, ip, nm, gw, GOAL_FALSE);
    if (GOAL_RES_ERR(res)) {
        goal_logErr("Set IP failed");
        return res;                                                                          ③
    }

    appl_modbusInit();                                                                       ④

    return GOAL_OK;
}
```

①LED の初期状態を設定します。

②使用するネットワークメディアアダプタ（MA）をオープンします。

③TCP/IP プロトコルで使用する IP アドレスを設定します。

④Modbus スタック部の初期化を行います。

(3) tcpCallback

本関数では、発生した TCP 通信イベントに対する処理を行います。

```
static void tcpCallback(
・・・
)
{
    GOAL_NET_ADDR_T remote;                    /* remote address */
    GOAL_STATUS_T res;                         /* result */

    if (cbType == GOAL_NET_CB_NEW_SOCKET) {
・・・
        /* check connectable IP address */
        res = r_modbus_chk_connectable_ip(remote.remoteIp);
        if (GOAL_RES_ERR(res)) {                                                    ①
・・・
            return;
        }

        /* check multiple connection */
        res = r_modbus_tcp_multi_connection();
        if (GOAL_RES_ERR(res)) {                                                    ②
・・・
            return;
        }

        res = r_modbus_tcp_multi_connection(pChan);
        if (GOAL_RES_ERR(res)) {
            /* Close oldest connection if new connection cannot be registered */
            res = goal_maChanTcpClose(pMaTcpHdl, appl_modbusTcpGetOldestConnection());
            if (GOAL_RES_ERR(res)) {
                goal_logErr("Failed to get Remote Address for socket %p", (void *) pChan);
            }                                                                       ③
            r_modbus_tcp_del_connection_list(r_modbus_tcp_get_oldest_connection());
            /* regist new connection */
            res = r_modbus_tcp_reg_connection_list(pChan);
            if (GOAL_RES_ERR(res)) {
                goal_logErr("Failed to get Remote Address for socket %p", (void *) pChan);
            }
        }
    }
    else if (cbType == GOAL_NET_CB_NEW_DATA) {
        goal_logInfo("Data received on tcp socket %p", (void *) pChan);

        res = appl_parseModbusPacket(pMaTcpHdl, pChan, pBuf);
        if (GOAL_RES_ERR(res)) {
            /* Close oldest connection if new connection cannot be registered */
            res = goal_maChanTcpClose(pMaTcpHdl, pChan);
            if (GOAL_RES_ERR(res)) {                                                ④
                goal_logErr("Failed to get Remote Address for socket %p", (void *) pChan);
            }
            r_modbus_tcp_del_connection_list(r_modbus_tcp_get_oldest_connection());
        }
    }
    else if (cbType == GOAL_NET_CB_CLOSING) {
        goal_logInfo("Closing TCP socket %p", (void *) pChan);

        r_modbus_tcp_del_connection_list(pChan);

        res = goal_maChanTcpGetRemoteAddr(pMaTcpHdl, pChan, &remote);
        if (GOAL_RES_ERR(res)) {                                                    ⑤
            goal_logErr("Failed to get Remote Address for socket %p", (void *) pChan);
        }
    }
}
```

① 接続要求をした IP アドレスからの接続を受け入れるかどうかをチェックします。対象の IP アドレスが接続不可の場合、要求のあったインスタンスをクローズします。

② 複数のクライアント接続を受け入れるかどうかをチェックします。マルチクライアント機能が無効かつ 2 つ目以降の接続要求だった場合は、新しい接続要求の TCP インスタンスをクローズします。

③ 新たに開かれた TCP/IP 接続のネットチャネルを、Modbus スタックに登録します。同時接続できる上限数は Modbus スタック側で管理しているため、必要に応じてネットチャネルの切断処理を行います。

④ 受信した TCP パケットを Modbus パケット解析処理に引き渡します。

⑤ 接続が閉じられたネットチャネルの通知を受け、Modbus スタックの管理から除外します。

(4) _appl_setupTcpSocket

本関数は、Modbus スタックから TCP ソケット生成時に呼び出されます。

```
GOAL_STATUS_T _appl_setupTcpSocket(uint32_t additionalPort)
{
    GOAL_STATUS_T res;                          /* result */
    GOAL_NET_ADDR_T addr;                       /* net address */
    GOAL_NET_CHAN_T *pChan;                     /* channel */
    uint32_t clients;

    res = goal_maChanTcpOpen(GOAL_NET_ID_DEFAULT, &pMaTcp);
    if (GOAL_RES_ERR(res)) {                                                    ①
        goal_logErr("error getting tcp MA");
        return res;
    }

    for (clients = 0; clients < MAXIMUM_NUMBER_OF_CLIENTS + 1; clients++) {
        /* register TCP server */                                               ②
        GOAL_MEMSET(&addr, 0, sizeof(GOAL_NET_ADDR_T));
        addr.localPort = (uint16_t) (TCP_PORT_NUMBER);
        res = goal_maChanTcpNew(pMaTcp, &pChan, NULL, &addr, GOAL_NET_TCP_LISTENER, tcpCallback);
        if (GOAL_OK != res) {
            goal_logErr("error while opening TCP server channel on port %"FMT_u32, (uint32_t) TCP_PORT_NUMBER);
            return res;
        }

        /* set TCP channel to non-blocking */
        res = goal_maChanTcpSetNonBlocking(pMaTcp, pChan, GOAL_TRUE);
        if (GOAL_OK != res) {
            goal_logErr("error while setting TCP channel to non-blocking");
            return res;
        }
    }

    /* greet */
    goal_logInfo("waiting for TCP connections on port %"FMT_u32"(n=%"FMT_u32")", TCP_PORT_NUMBER, clients);

    if (0 != additionalPort)
    {
        res = goal_maChanTcpOpen(GOAL_NET_ID_DEFAULT, &pMaTcpAdditional);
        if (GOAL_RES_ERR(res)) {                                                ③
            goal_logErr("error getting tcp MA");
            return res;
        }

        for (clients = 0; clients < MAXIMUM_NUMBER_OF_CLIENTS + 1; clients++)
        {                                                                       ④
            GOAL_MEMSET(&addr, 0, sizeof(GOAL_NET_ADDR_T));
            addr.localPort = (uint16_t) (additionalPort);
            res = goal_maChanTcpNew(pMaTcpAdditional, &pChan, NULL, &addr, GOAL_NET_TCP_LISTENER, tcpCallback);
・・・
        }

        /* greet */
        goal_logInfo("waiting for TCP connections on port %"FMT_u32"(n=%"FMT_u32")", additionalPort, clients);
    }

    return res;
}
```

①TCP チャネルメディアアダプタ（MA）をオープンします。

②通常の接続ポート（**TCP_PORT_NUMBER**）用の TCP チャネルを（**MAXIMUM_NUMBER_OF_CLIENTS**＋１）個生成します。TCP チャネルはノンブロッキングモードで準備します。

③追加接続ポート用の TCP チャネルメディアアダプタ（MA）をオープンします。

④追加接続ポート（**TCP_ADDITIONAL_PORT_NUMBER**）用の TCP チャネルを（**MAXIMUM_NUMBER_OF_CLIENTS**＋１）個生成します。

本関数では、**MAXIMUM_NUMBER_OF_CLIENTS** で指定された上限数＋１分の uGOAL インスタンスを作成します。**MAXIMUM_NUMBER_OF_CLIENTS** の上限数を増やす場合は、uGOAL としてのリソース（**CONFIG_UGOAL_HEAP_BUFFER_SIZE** など）も増やす必要がありますのでご注意ください。

3.1.2 Modbus スタック構成部

Modbus スタックは、uGOAL の TCP/IP 機能と連動して機能を実現しています。

3.1.2.1 コンフィグ設定

ユーザが変更できる Modbus スタックの設定を表 3-1 および表 3-2 に示します。各設定は r_modbus_tcp_cfg.h および r_modbus_serial_cfg.h にマクロで定義されています。

表 3-1 ModbusTCP コンフィグ設定

#	マクロ名	デフォルト	内容
1	**TCP_PORT_NUMBER**	502	Modbus 接続として受け付ける TCP ポートを指定します。
2	**TCP_ADDITIONAL_PORT_NUMBER**	1024	TCP_PORT_NUMBER で設定したポートに追加で受け付けるポートを指定します。0 に設定すると追加ポートは無効となります。
3	**MAXIMUM_NUMBER_OF_CLIENTS**	7	Modbus スタックで接続できる TCP クライアントの上限を指定します。
4	**MBAPP_INIT_MULTIPLE_CLIENT_FLAG**	有効	マルチクライアント機能の有効・無効を指定する。 有効:**ENABLE_MULTIPLE_CLIENT_CONNECTION** 無効:**DISABLE_MULTIPLE_CLIENT_CONNECTION**
5	**MBAPP_INIT_IP_TABLE_FLAG**	DISABLE	IP アドレスのリスト機能の有効・無効を指定する。設定値は、ENABLE/DISABLE。
6	**MBAPP_INIT_IP_TABLE_MODE**	ACCEPT	IP アドレスのリスト機能の許可・除外機能を指定する。設定値は ACCEPT/REJECT。

表 3-2　Modbus シリアルコンフィグ設定

#	マクロ名	デフォルト	内容
1	**MBAPP_INIT_GW_SERIAL_MODE**	RTU モード	Modbus シリアルデバイスとの通信方式を指定する。 RTU モード:**MODBUS_RTU_MASTER_MODE** ASCII モード:**MODBUS_ASCII_MASTER_MODE**
2	**MBAPP_INIT_SLAVE_ID**	1	Modbus シリアルデバイスのデバイス ID を設定する。設定値は 1～247。
3	**MBAPP_SERIAL_BAUDRATE**	115200	シリアル通信の通信速度を bps 単位で指定する。
4	**MBAPP_SERIAL_PARITY**	パリティなし	シリアル通信のパリティ設定を指定する。設定値はシリアル通信のドライバ I/F に依存します。
5	**MBAPP_SERIAL_STOPBITS**	１ビット	シリアル通信のストップビット設定を指定する。設定値はシリアル通信のドライバ I/F に依存します。
6	**MBAPP_INIT_RESPONSE_TIMEOUT_MS**	2000	シリアル通信の応答待ちのタイムアウトをミリ秒単位で指定する。
7	**MBAPP_INIT_TURNAROUND_DELAY_MS**	200	ブロードキャスト送信時に連続送信する際の遅延をミリ秒単位で指定する。
8	**MBAPP_INIT_RETRY_COUNT**	3	シリアル通信の最大リトライ回数を指定。 整数値での指定。

3.1.2.2 スタック初期化

各種初期化を実行し、Modbus スタックを開始します。これらの初期化は uGOAL の初期化の後に行われる必要があります。以下、初期化で行う処理について説明します。尚、本節で説明している機能は、サンプルソフト上では関数 *appl_initModbus* にて実装されています。

1) 各ファンクションコードに対応したコールバック関数の登録

クライアントからの各ファンクションコードに対する処理要求をユーザ定義の関数に関連付けます。Modbus スタックは要求を受信した際、登録された関数を呼び出します。

登録設定はスタック API（*r_modbus_slave_map_init*）で行います。スタック API に設定する引数の詳細を以下に示します。下記テーブルに、対応するコールバック関数を設定し、スタック API を実行することでスタック内に登録されます。

サンプルプログラムでは、表 3-3 に示す関数がデフォルト関数として登録されています。

表 3-3 ファンクションコードに対応するデフォルトコールバック関数一覧

Code	Function Name	Default callback function
1	Read Coils	cb_func_code01
2	Read Discrete Inputs	cb_func_code02
3	Read Holding Registers	cb_func_code03
4	Read Input Registers	cb_func_code04
5	Write Single Coil	cb_func_code05
6	Write Single Register	cb_func_code06
15	Write Multiple Coils	cb_func_code15
16	Write Multiple Registers	cb_func_code16
23	Read/Write Multiple Registers	cb_func_code23

・ファンクションコードマッピングテーブル (*slave_map_init_t*)

```
typedef struct _slave_map_init{
    fp_function_code1_t     fp_function_code1;     /* function code-1 Read coils ポインタ */
    fp_function_code2_t     fp_function_code2;     /* function code-2 Read discrete inputs ポインタ */
    fp_function_code3_t     fp_function_code3;     /* function code-3 Read holding registers ポインタ */
    fp_function_code4_t     fp_function_code4;     /* function code-4 Read input registers ポインタ */
    fp_function_code5_t     fp_function_code5;     /* function code-5 Write single coil ポインタ */
    fp_function_code6_t     fp_function_code6;     /* function code-6 Write single register ポインタ */
    fp_function_code15_t    fp_function_code15;    /* function code-15 Write multiple coils ポインタ */
    fp_function_code16_t    fp_function_code16;    /* function code-16 Write multiple registers ポインタ */
    fp_function_code23_t    fp_function_code23;    /* function code-23 Read/Write multiple registers ポインタ*/
}slave_map_init_t, *p_slave_map_init_t;
```

2) Modbus スタックの初期化

Modbus スタックの初期化をスタック API（*r_modbus_tcp_init_stack* または *r_modbus_tcp_init_gateway_stack*）で行います。当該 API 経由で、*appl_setupTcpSocket* が呼び出され、必要な TCP ソケットが生成されます。

3.1.2.3 ファンクションコード対応

本サンプルで使用している Modbus スタックは、ユーザによる任意の Modbus データモデルの設計をサポートするために、それらの Modbus データモデルにアクセスするファンクションコード処理を、コールバック関数として実装します。サンプル上では、*function_code.c* に含まれています。

以下にファンクションコード実装時の要点を示します。

- Modbus プロトコルスタックは、表 2-5 のファンクションコードに対応するコールバック関数を登録できます。本サンプルでは、全てのコールバック関数の実装例を示しています。
- Modbus プロトコルは固有のデータモデルを持ち、データモデルは 4 つのデータ型と、各データ型に対応するアドレス空間で構成されます。本プロトコルスタックが対応する Modbus データ型を表 2-6 に示します。
- ファンクションコードを処理するコールバック関数は、各々対応するデータ型にアクセスする処理が要求されます。
- Modbus プロトコルにおいて、スレーブは各データ型に対して最大 65536（0x10000）個のデータを持つことができ、それぞれ 1~65536(0x00001~0x10000)の範囲で参照アドレスを割り振る事が出来ます。
- また、参照アドレスは任意の物理アドレスを参照することが出来ます。

表 3-4 Modbus ファンクションコード

Code	Code (Hex)	Function Name	Description
1	1h	Read Coils	複数の Coil アドレスのデータを読み出す
2	2h	Read Discrete Inputs	複数の Discrete Inputs アドレスのデータを読み出す
3	3h	Read Holding Registers	複数の Holding Registers アドレスのデータを読み出す
4	4h	Read Input Registers	複数の Input Registers アドレスのデータを読み出す
5	5h	Write Single Coil	単一の Coil アドレスにデータを書き込む
6	6h	Write Single Register	単一の Holding Register アドレスにデータを書き込む
15	Fh	Write Multiple Coils	複数の Coil アドレスにデータを書き込む
16	10h	Write Multiple Registers	複数の Holding Register アドレスにデータを書き込む
23	17h	Read/Write Multiple Registers	複数の Holding Register アドレスにデータを書き込んだ後、他の複数の Holding Register アドレスのデータを読み出す

表 3-5 Modbus データ型

Name	Bits	Type of access	Description
Discrete Inputs	1	Read	I/O システムによって提供できるデータの型
Coils	1	Read/Write	アプリケーションプログラムによって変更できるデータ型
Input Registers	16	Read	I/O システムによって提供できるデータの型
Holding Registers	16	Read/Write	アプリケーションプログラムによって変更できるデータ型

Modbus ファンクションコードのコールバック関数ですべき処理は以下の 2 つとなります。

1. ポインタ引数のリクエスト構造体を参照して、ポインタ引数のレスポンス構造体にファンクションコードの処理結果をセットする。
2. 設計された Modbus データモデルが持たない参照アドレスが参照された場合、例外コード 0x02 をレスポンス構造体にセットする。

また、ファンクションコード処理中に発生した回復不能なエラーに対して、例外コード 0x04 をセットすることができます。

表 3-6 例外コード

Code	Code (Hex)	Function Name	Description
2	2h	Illegal Data Address	指定アドレス値、もしくは指定アドレス範囲に、設計された Modbus データモデルにとって不正なアドレスが含まれる場合に返す。
4	4h	Slave Device Failure	ファンクションコードの処理中に回復不能なエラーが発生した場合に返す。

各ファンクションコードに対応したコールバック関数は、以下の書式で定義されます。各コールバック関数の引数に使用されている構造体の詳細については Appendix.A を参照してください。

【書式】

uint32_t (*fp_function_code<function code>_t)(p_req_<function code 名>_t pt_request,
p_resp_<function code 名>_t pt_response);

【引数】		
p_req_<function code 名>_t	pt_request	ファンクションコード要求情報が格納された構造体へのポインタ
p_resp_<function code 名>_t	pt_response	ファンクションコード応答データを格納する構造体へのポインタ

【戻り値】	
uint32_t	0 : 正常,1 : 異常

以下に Read Coil（ファンクションコード 01h）実行時に呼び出されるコールバック関数のサンプルコード例を示します。

例) ファイル: *07_mbus_tcp_server\mbapp_slave\function_code.c*

```
uint32_t cb_func_code01(p_req_read_coils_t  pt_request,
                        p_resp_read_coils_t pt_response)
{
    uint8_t     u8_data;
    int         i;

    /* Copy invariant fields between response and request. */
    pt_response->u16_transaction_id = pt_request->u16_transaction_id;
    pt_response->u16_protocol_id    = pt_request->u16_protocol_id;
    pt_response->u8_slave_id        = pt_request->u8_slave_id;

    /* Check starting address and ending address. ( START <= address < END ) */
    pt_response->u8_exception_code = check_illegal_coils_address( pt_request->u16_start_addr,
                                                                  pt_request->u16_num_of_coils );
    if ( ERR_OK != pt_response->u8_exception_code ){ return ERR_OK; }

    /* Get response data by accessing address */
    for( i = 0; i < pt_request->u16_num_of_coils; i++ )
    {
        /* Get response data by accessing address. */
        pt_response->u8_exception_code = read_coil_address(pt_request->u16_start_addr+i, &u8_data);
        if ( ERR_OK != pt_response->u8_exception_code ){ return ERR_OK; }   /* Return value is ignored on stack. */

        /* Set a bit into a response byte which are cleared every byte. */
        if( i % 8 == 0 ) { pt_response->aru8_data[i/8] = 0; }
        pt_response->aru8_data[i/8] |= u8_data << ( i % 8 );
    }

    /* Calculate the number of bytes of response data. */
    pt_response->u8_num_of_bytes = _num_of_bytes(pt_request->u16_num_of_coils);

    return ERR_OK; // Return value is ignored on stack.
}
```

1. 応答パケットのヘッダ部に格納する情報を設定します。
2. 指示されたアドレスとデータ長からアクセス範囲に異常がないかをチェックします。
3. Coils 領域から指定されたアドレス範囲のデータを取得し、応答パケットに格納します。

3.1.2.4 TCP/IP 通信管理

本節では、Modbus スタック上の TCP/IP 通信に関連する実装について説明します。

ホワイトリスト・ブラックリスト機能

IP アドレスのホワイトリストおよびブラックリスト機能を使用する場合、対象となる IP アドレスを登録する必要があります。IP アドレスの登録は、r_modbus_tcp_add_ip_addr にて行います。
サンプルプログラムでは、r_modbus_enable_host_ip にてスタック初期化前に登録を行っています。

3.1.3 シリアル通信部

本節では、Modbus TCP/シリアルゲートウェイモード使用時に Modbus シリアルデバイスとの通信を行う部分に関連する実装について説明します。本節で説明する内容は、各サンプルアプリケーションの plat_modbus.c にて定義されています

1) シリアル通信制御

本サンプルでは、シリアルデバイスとの通信に FIT（Firmware Integration Technology）のシリアル通信インターフェース（SCI）モジュールを使用しています。シリアル通信は、Modbus スタック構成部から呼び出されるため、Modbus スタックとの I/F に以下の API が設定されています。

#	API 名	機能
1	r_modbus_serial_open	シリアル通信をオープンする。
2	r_modbus_serial_read	シリアルデータを受信する。
3	r_modbus_serial_write	シリアルデータを送信する。
4	r_modbus_rs485_tx_enable	シリアル送信許可状態にする。
5	r_modbus_rs485_tx_disable	シリアル送信禁止状態にする。

(2) RTU/ASCII モード RS485 通信タイミング制御

RTU モードの場合、少なくとも 3.5 文字分の無通信時間で始まり、3.5 文字分の無通信時間で終了します。本サンプルでは、この無通信時間の測定に FIT のコンペアマッチタイマ（CMT）モジュールを使用しています。無通信時間の測定は、Modbus スタック構成部で行われるため、Modbus スタックとの I/F に以下の API が設定されています。

#	API 名	機能
1	r_modbus_init_timer	測定に使用するタイマを初期化する。
2	r_modbus_timer_oneshot	時間測定開始する。
3	r_modbus_timer_stop	タイマを停止する。

4. 評価用ツールを用いた通信テスト

Modbus クライアント評価用ツール（ModbusDemoApplication.exe）を使用して、Modbus プロトコルスタックサンプルプログラム動作を確認することが可能です。

評価用ツール ModbusDemoApplication:
r18an0064**xxx \ tool \ **ModbusDemoApplication.zip**

4.1 Modbus TCP サーバ接続

(1)接続構成

SEMB1320 評価ボードを Modbus TCP サーバとして動作する場合の構成を示します。

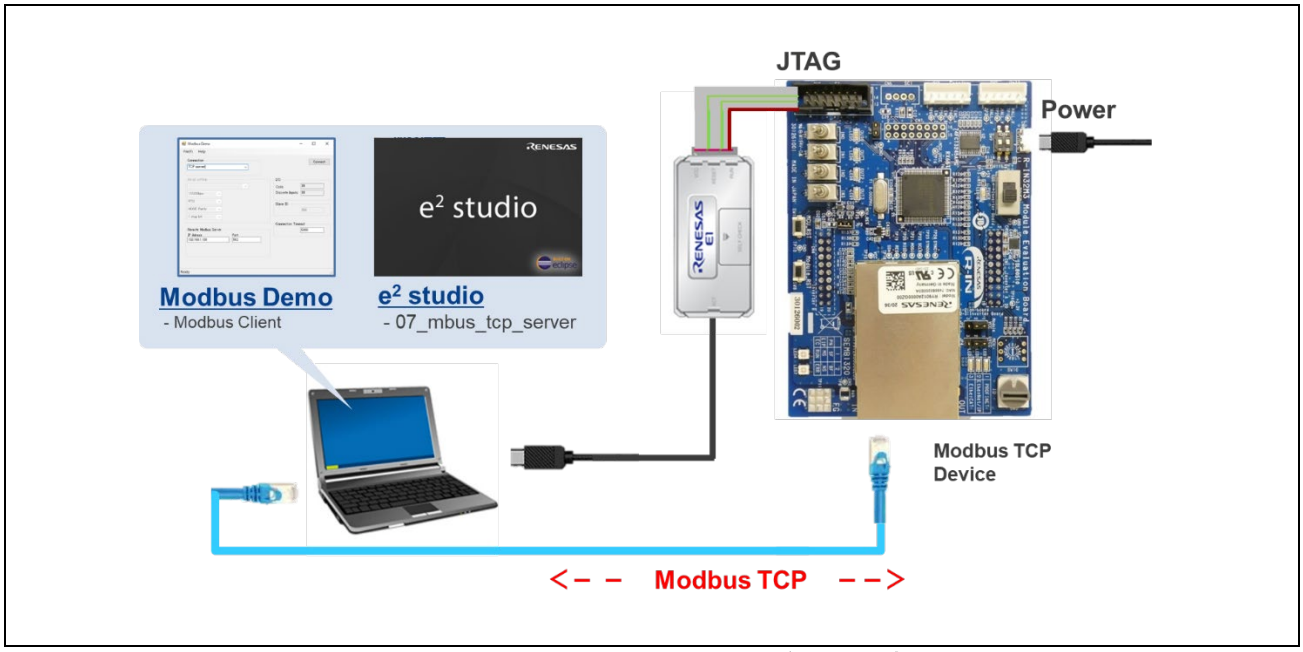


図 4-1 Modbus TCP サーバ評価環境

[注意] Modbus 実行時は、SEMB1320 のプロトコル LED (LED1, LED2, LED3)は全消灯となります。

(2) 評価用ツールの動作概要

評価用ツールは Modbus クライアントとして動作します。

SEMB1320 評価ボードは、評価用ツールとの Modbus 通信に応じて SW の状態の送信、もしくは LED の点灯の制御を行います。LED に対応する Coil アドレスを表 2-2 に、SW に対応する Discrete Input アドレスを表 2-3 に示します。

表 4-1 Coil アドレスと対応する LED

Coil アドレス	対応する LED
0001h	LED5
0002h	LED6
0003h	LED8
0004h	LED9

表 4-2 Discrete Input アドレスと対応する SW

Discrete Input アドレス	対応する SW
0001h	SW2
0002h	SW4
0003h	SW5
0004h	SW6

(3) 評価用ツールの使用方法

“Connection”で動作モードを選択し、各種パラメータを設定してください。

- **Connection**
  - 「TCP server」を選択します。

- **Serial setting**
  - 使用しません。

- **Remote Modbus Server**
  - デバイスに合わせて IP アドレス、ポート番号を設定します。

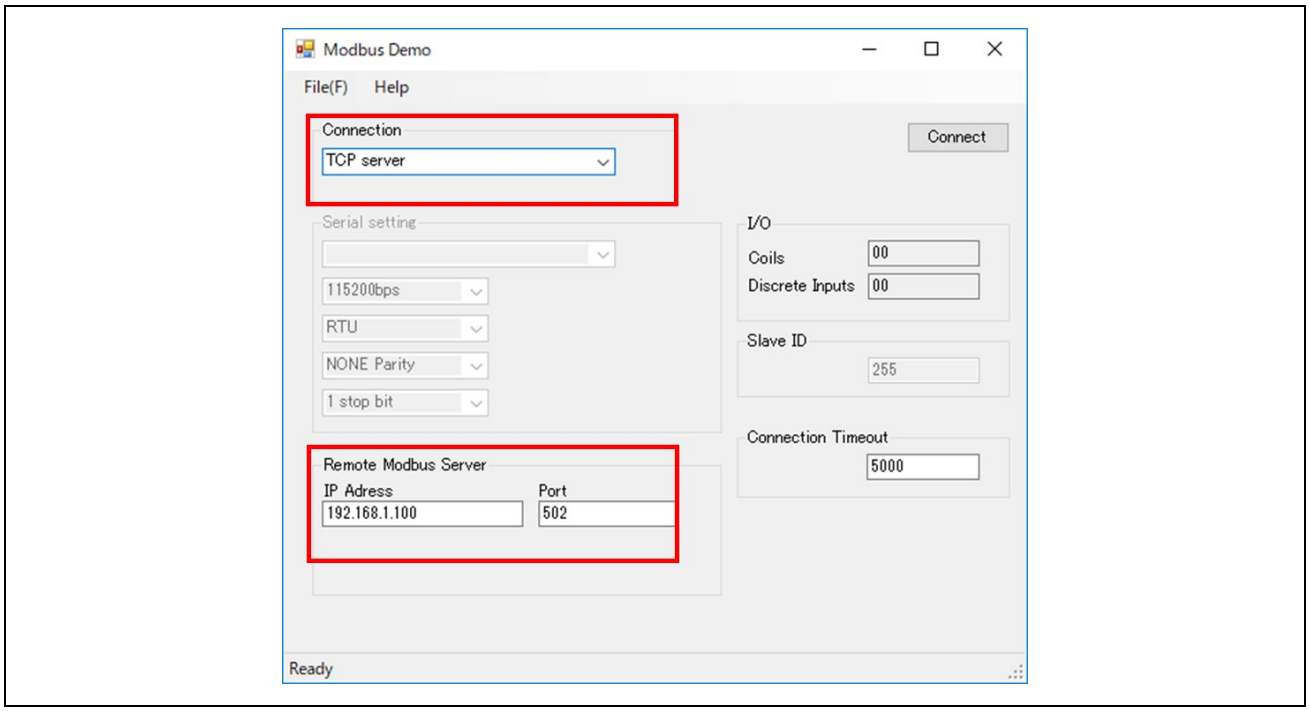


図 4-2 Modbus Demo tool 設定 [TCP server]

（次ページへ続く）

サンプルアプリケーションが Modbus TCP サーバスタックモード時、評価用ツールは Modbus クライアントとして動作し、以下の制御となります。

- [Connect] を実行すると評価用ツールは、Read Discrete Inputs リクエストを送信します。
- Modbus TCP サンプルアプリケーションは、Read Discrete Inputs リクエストを受信し、SEMB1320 評価ボード SW (SW2, SW4, SW5, SW6) の状態が Discrete Inputs に反映されます。
- 評価用ツールは、Read Discrete Inputs リクエストのレスポンスとして受信した SW の状態に応じて、SEMB1320 評価ボードの LED の更新方法を決定し、Write Multiple Coils リクエストを送信します。
  - **SW “0”の場合**：Coils テキストボックスの値が 01→02→04→08→01…の順に値が変化
    その Coils の値が LED(LED5, LED6, LED8 ,LED9)に反映されます。
  - **SW “0 以外”の場合**：Coils テキストボックスに任意の値を入力できます。
    その Coils の値が LED(LED5, LED6, LED8 ,LED9)に反映されます。

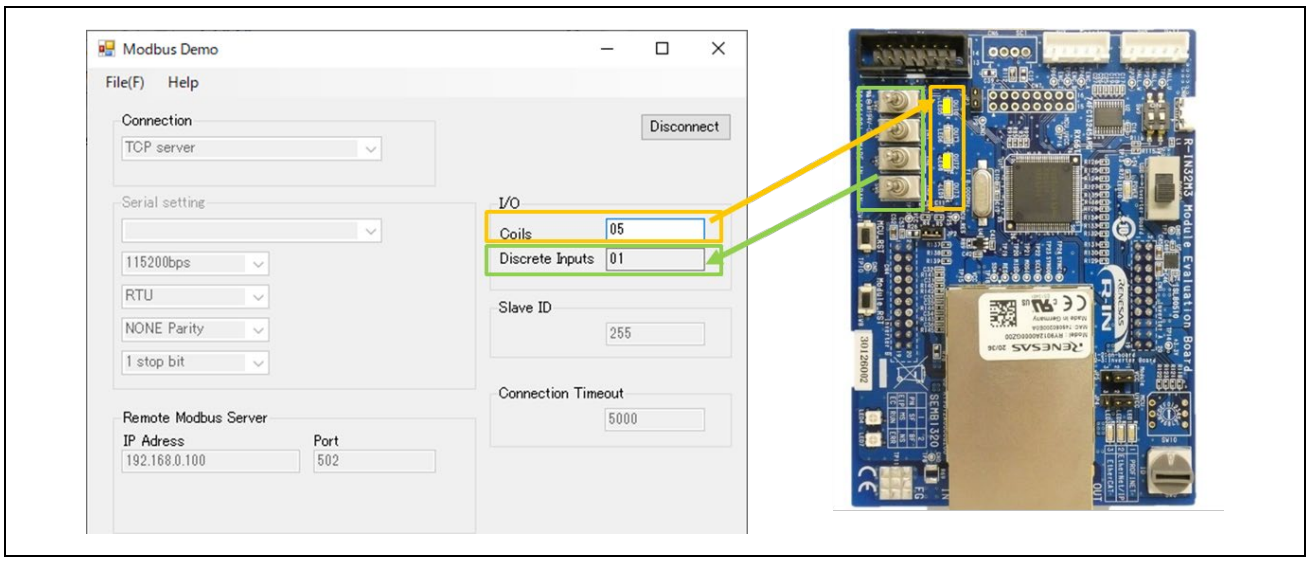


図 4-3　Modbus クライアントとして動作する評価ツール

4.2 Modbus TCP ゲートウェイ接続

(1) 接続構成

SEMB1320 評価ボードを Modbus TCP ゲートウェイとして動作する場合の構成を示します。

弊社では、RX72M 通信ボード[Tessera Technology 製 TS-TCS07298] を Modbus RTU/ASCII スレーブデバイスとしてシリアル接続 (RS-485: 半二重通信 (Half Duplex)) した環境で評価しています。RX72M 通信ボードおよび、Modbus RTU/ASCII スレーブデバイス用プログラムについては、『RX72M Group 通信ボード Modbus スタートアップマニュアル(R01AN4862****)』を参照ください。

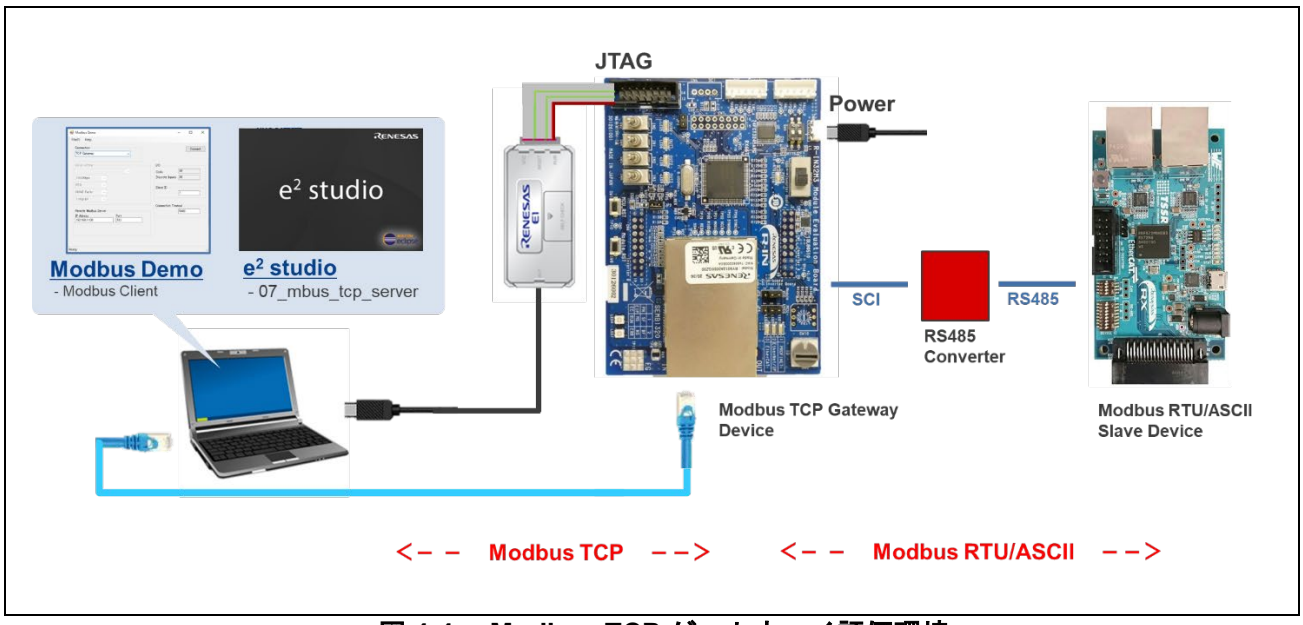


図 4-4 Modbus TCP ゲートウェイ評価環境

[注意] Modbus サンプルアプリ実行時は、プロトコル LED (LED1, LED2, LED3)は全消灯となります。

(2) 評価用ツールの動作概要

Modbus TCP の場合と同様、Modbus Demo tool は Modbus クライアントとして動作します。ただし、評価用ツールからの指示は SEMB1320 評価ボードにシリアル接続された Modbus RTU/ASCII スレーブデバイスに対して行われることになるため、その動作は対象デバイスに依存します。

一例としてシリアルスレーブとして使用している RX72M 通信ボードむけサンプルソフト [an-r01an4862xx****-rx72m-modbus]との組合せでは、評価用ツール Coils への設定値で RX72M 評価ボード上の汎用 LED1-4 が点灯します。

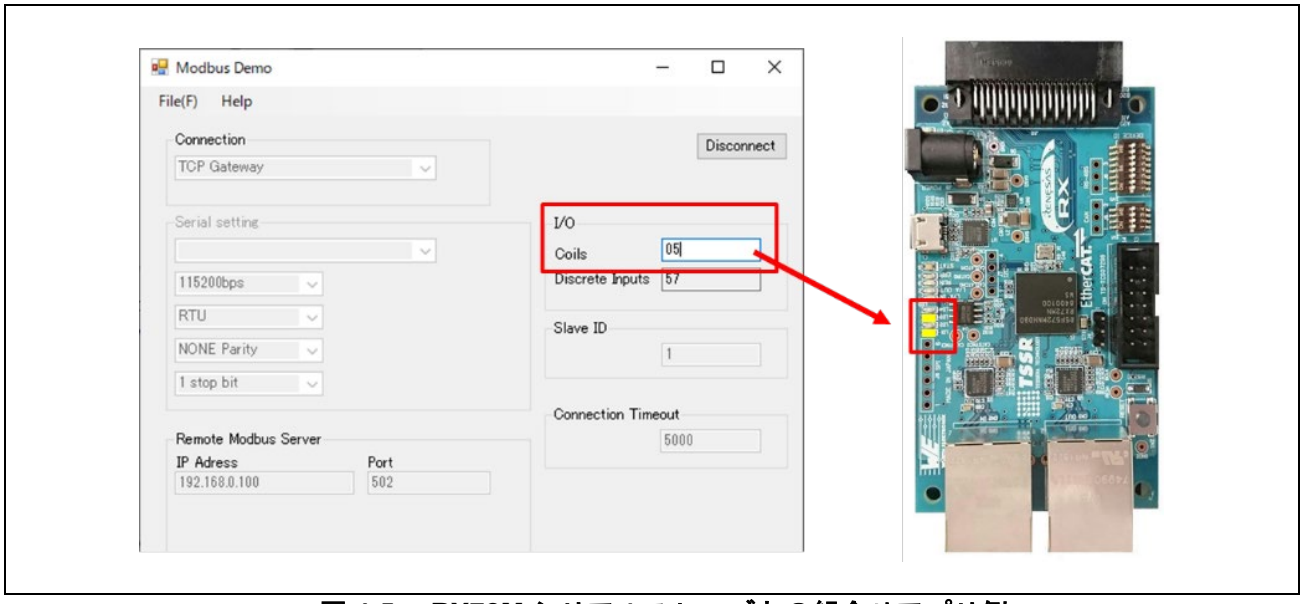


図 4-5 RX72M シリアルスレーブとの組合せアプリ例

(3) 評価用ツールの使用方法

“Connection”で動作モードを選択し、各種パラメータを設定してください。

- **Connection**
  - ➢ TCP Gateway を選択します。

- **Serial setting**
  - ➢ 使用しません。

- **Remote Modbus Server**
  - ➢ デバイスに合わせて IP アドレス、ポート番号を設定します。

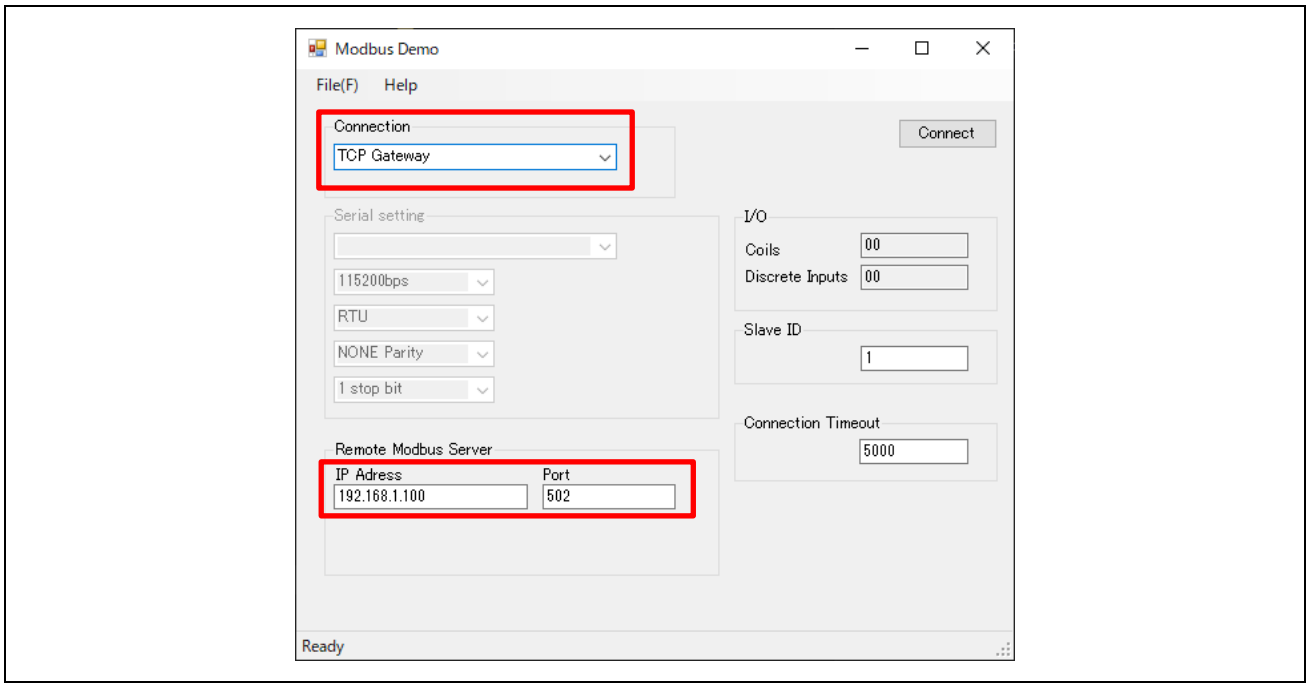


図 4-6 Modbus Demo tool 設定 [TCP Gateway]

サンプルアプリケーションが Modbus TCP ゲートウェイ モード時、評価用ツールは Modbus クライアントとして動作し、以下の制御となります。

- [Connect] を実行すると評価用ツールは、Read Discrete Inputs および Write Multiple Coils リクエストを送信します。
- RX66T の Modbus TCP Gaetway サンプルアプリケーションは、Read Discrete Inputs または Write Multiple Coils リクエストを受信し、Modbus シリアルスレーブ デバイスの RX72M communication board に対して、同じ内容の Read Discrete Inputs または Write Multiple Coils リクエストを発行します。
- リクエスト発行後、Modbus シリアルスレーブ デバイスからのレスポンスパケットを受信し、リクエスト時とは逆にクライアントに対して、同じ内容の Read Discrete Inputs または Write Multiple Coils レスポンスを発行します。

Appendix

A. ファンクションコード用構造体

Modbus スタックで登録する各ファンクションコード用コールバック関数に使用される引数の構造体定義について示します。

・Read coils 要求テーブル (req_read_coils_t)

```
typedef struct _req_read_coils{
    uint16_t    u16_transaction_id;     /* トランザクション ID 指定 */
    uint16_t    u16_protocol_id;        /* プロトコル ID 指定 */
    uint8_t     u8_slave_id;            /* スレーブ ID */
    uint16_t    u16_start_addr;         /* リードする coil の開始アドレス */
    uint16_t    u16_num_of_coils;       /* リードする coil の個数 */
}req_read_coils_t, *p_req_read_coils_t;
```

・Read coils 応答テーブル (resp_read_coils_t)

```
struct _resp_read_coils{
    uint16_t    u16_transaction_id;     /* トランザクション ID 指定 */
    uint16_t    u16_protocol_id;        /* プロトコル ID 指定 */
    uint8_t     u8_slave_id;            /* スレーブ ID（自局 ID） */
    uint8_t     u8_exception_code;      /* 要求に対してエラーを検出した場合に 0 以外を設定。
                                           正常終了の場合は 0 を設定。0 以外が設定された場合、
                                           aru8_data は無効となります。 */
    uint8_t     u8_num_of_bytes;        /* リードしたデータのバイト数 */
    uint8_t     aru8_data[MAX_DISCRETE_DATA]; /* リードデータ */
}resp_read_coils_t, *p_resp_read_coils_t;
```

・Read inputs 要求テーブル(req_read_inputs_t)

```
typedef struct _req_read_inputs{
    uint16_t    u16_transaction_id;     /* トランザクション ID 指定 */
    uint16_t    u16_protocol_id;        /* プロトコル ID 指定 */
    uint8_t     u8_slave_id;            /* スレーブ ID */
    uint16_t    u16_start_addr;         /* リードする discrete inputs の開始アドレス */
    uint16_t    u16_num_of_inputs;      /* リードするレジスタの個数 */
}req_read_inputs_t, *p_req_read_inputs_t;
```

・Read inputs 応答テーブル (resp_read_inputs_t)

```
typedef struct _resp_read_inputs{
    uint16_t    u16_transaction_id;          /* トランザクション ID 指定 */
    uint16_t    u16_protocol_id;             /* プロトコル ID 指定 */
    uint8_t     u8_slave_id;                 /* スレーブ ID */
    uint8_t     u8_exception_code;           /* 要求に対してエラーを検出した場合に 0 以外を設定。
                                                正常終了の場合は 0 を設定。0 以外が設定された場合、
                                                aru8_data は無効となります。 */
    uint8_t     u8_num_of_bytes;             /* リードしたデータのバイト数 */
    uint8_t     aru8_data[MAX_DISCRETE_DATA]; /* リードデータ */
}resp_read_inputs_t, *p_resp_read_inputs_t;
```

・Read holding registers 要求テーブル(req_read_holding_reg_t)

```
typedef struct _req_read_holding_reg{
    uint16_t    u16_transaction_id;          /* トランザクション ID 指定 */
    uint16_t    u16_protocol_id;             /* プロトコル ID 指定 */
    uint8_t     u8_slave_id;                 /* スレーブ ID */
    uint16_t    u16_start_addr;              /* リードする holding register の開始アドレス */
    uint16_t    u16_num_of_reg;              /* リードするレジスタの個数 */
}req_read_holding_reg_t, *p_req_read_holding_reg_t;
```

・Read holding registers 応答テーブル(resp_read_holding_reg_t)

```
typedef struct _resp_read_holding_reg{
    uint16_t    u16_transaction_id;          /* トランザクション ID 指定 */
    uint16_t    u16_protocol_id;             /* プロトコル ID 指定 */
    uint8_t     u8_slave_id;                 /* スレーブ ID */
    uint8_t     u8_exception_code;           /* 要求に対してエラーを検出した場合に 0 以外を設定。正常
                                                終了の場合は 0 を設定。0 以外が設定された場合、
                                                aru16_data は無効となります。 */
    uint8_t     u8_num_of_bytes;             /* リードしたデータのバイト数 */
    uint16_t    aru16_data[MAX_REG_DATA];    /* リードデータ */
}resp_read_holding_reg_t, p_resp_read_holding_reg_t;
```

・Read input registers 要求テーブル (req_read_input_reg_t)

```
typedef struct _req_read_input_reg{
    uint16_t    u16_transaction_id;          /* トランザクション ID 指定 */
    uint16_t    u16_protocol_id;             /* プロトコル ID 指定 */
    uint8_t     u8_slave_id;                 /* スレーブ ID */
    uint16_t    u16_start_addr;              /* リードする input register の開始アドレス */
    uint16_t    u16_num_of_reg;              /* リードするレジスタの個数 */
}req_read_input_reg_t, *p_req_read_input_reg_t;
```

・Read input registers 応答テーブル(resp_read_input_reg_t)

```
typedef struct _resp_read_input_reg{
    uint16_t    u16_transaction_id;     /* トランザクション ID 指定 */
    uint16_t    u16_protocol_id;        /* プロトコル ID 指定 */
    uint8_t     u8_slave_id;            /* スレーブ ID */
    uint8_t     u8_exception_code;      /* 要求に対してエラーを検出した場合に 0 以外を設定。正常
                                           終了の場合は 0 を設定。0 以外が設定された場合、
                                           aru16_data は無効となります。 */
    uint8_t     u8_num_of_bytes;        /* リードしたデータのバイト数 */
    uint16_t    aru16_data[MAX_REG_DATA]; /* リードデータ */
}resp_read_input_reg_t, p_resp_read_input_reg_t;
```

・Write single coil 要求テーブル(req_write_single_coil_t)

```
typedef struct _req_write_single_coil
    uint16_t    u16_transaction_id;     /* トランザクション ID 指定 */
    uint16_t    u16_protocol_id;        /* プロトコル ID 指定 */
    uint8_t     u8_slave_id;            /* スレーブ ID */
    uint16_t    u16_output_addr;        /* ライトする coil のアドレス */
    uint16_t    u16_output_value;       /* ライトする値 */
}req_write_single_coil_t, *p_req_write_single_coil_t;
```

・Write single coil 応答テーブル(resp_write_single_coil_t)

```
typedef struct _resp_write_single_coil{
    uint16_t    u16_transaction_id;     /* トランザクション ID 指定 */
    uint16_t    u16_protocol_id;        /* プロトコル ID 指定 */
    uint8_t     u8_slave_id;            /* スレーブ ID */
    uint8_t     u8_exception_code;      /* 要求に対してエラーを検出した場合に 0 以外を設定。正常終
                                           了の場合は 0 を設定。 */
    uint16_t    u16_output_addr;        /* ライトしたアドレス */
    uint16_t    u16_output_value;       /* ライトしたデータ */
}resp_write_single_coil_t, *p_resp_write_single_coil_t;
```

・Write single register 要求テーブル(req_write_single_reg_t)

```
typedef struct _req_write_single_reg{
    uint16_t    u16_transaction_id;     /* トランザクション ID 指定 */
    uint16_t    u16_protocol_id;        /* プロトコル ID 指定 */
    uint8_t     u8_slave_id;            /* スレーブ ID */
    uint16_t    u16_register_addr;      /* ライトするレジスタのアドレス */
    uint16_t    u16_register_value;     /* ライトするデータ */
}req_write_single_reg_t, *p_req_write_single_reg_t;
```

・Write single register 応答テーブル(resp_write_single_reg_t)

```
typedef struct _resp_write_single_reg{
    uint16_t    u16_transaction_id;     /* トランザクション ID 指定 */
    uint16_t    u16_protocol_id;        /* プロトコル ID 指定 */
    uint8_t     u8_slave_id;            /* スレーブ ID */
    uint8_t     u8_exception_code;      /* 要求に対してエラーを検出した場合に 0 以外を設定。正常終了の
                                           場合は 0 を設定。 */
    uint16_t    u16_register_addr;      /* ライトしたレジスタのアドレス */
    uint16_t    u16_register_value;     /* ライトしたデータ */
}resp_write_single_reg_t, *p_resp_write_single_reg_t;
```

・Write multiple coils 要求テーブル(req_write_multiple_coils_t)

```
typedef struct _req_write_single_reg{
    uint16_t    u16_transaction_id;             /* トランザクション ID 指定 */
    uint16_t    u16_protocol_id;                /* プロトコル ID 指定 */
    uint8_t     u8_slave_id;                    /* スレーブ ID */
    uint16_t    u16_start_addr;                 /* ライトする coil の開始アドレス */
    uint16_t    u16_num_of_outputs;             /* ライトする coil の個数 */
    uint8_t     u8_num_of_bytes;                /* ライトするデータのバイト数 */
    uint8_t     aru8_data[MAX_DISCRETE_DATA];   /* ライトデータ */
}req_write_single_reg_t, *p_req_write_single_reg_t;
```

・Write multiple coils 応答テーブル(resp_write_multiple_coils_t)

```
typedef struct _resp_write_multiple_coils{
    uint16_t    u16_transaction_id;     /* トランザクション ID 指定 */
    uint16_t    u16_protocol_id;        /* プロトコル ID 指定 */
    uint8_t     u8_slave_id;            /* スレーブ ID */
    uint8_t     u8_exception_code;      /* 要求に対してエラーを検出した場合に 0 以外を設定。
                                           正常終了の場合は 0 を設定。 */
    uint16_t    u16_start_addr;         /* ライトした開始アドレス */
    uint16_t    u16_num_of_outputs;     /* ライトしたデータ */
}resp_write_multiple_coils_t, *p_resp_write_multiple_coils_t;
```

・Write multiple registers 要求テーブル(req_write_multiple_reg_t)

```
typedef struct _req_write_multiple_reg{
    uint16_t    u16_transaction_id;     /* トランザクション ID 指定 */
    uint16_t    u16_protocol_id;        /* プロトコル ID 指定 */
    uint8_t     u8_slave_id;            /* スレーブ ID */
    uint16_t    u16_start_addr;         /* ライトするレジスタの開始アドレス */
    uint16_t    u16_num_of_reg;         /* ライトするレジスタの個数 */
    uint8_t     u8_num_of_bytes;        /* ライトするデータのバイト数 */
    uint16_t    aru16_data[MAX_REG_DATA];   /* ライトデータ */
}req_write_multiple_reg_t, *p_req_write_multiple_reg_t;
```

・Write multiple registers 応答テーブル(resp_write_multiple_reg_t)

```
typedef struct _resp_write_multiple_reg{
    uint16_t    u16_transaction_id;     /* トランザクション ID 指定 */
    uint16_t    u16_protocol_id;        /* プロトコル ID 指定 */
    uint8_t     u8_slave_id;            /* スレーブ ID */
    uint8_t     u8_exception_code;      /* 要求に対してエラーを検出した場合に 0 以外を設定。正常終
                                           了の場合は 0 を設定。 */
    uint16_t    u16_start_addr;         /* ライトした開始アドレス */
    uint16_t    u16_num_of_reg;         /* ライトしたレジスタの個数 */
}resp_write_multiple_reg_t, *p_resp_write_multiple_reg_t;
```

・Read/Write multiple registers 要求テーブル(req_read_write_multiple_reg_t)

```
typedef struct _req_read_write_multiple_reg{
    uint16_t    u16_transaction_id;     /* トランザクション ID 指定 */
    uint16_t    u16_protocol_id;        /* プロトコル ID 指定 */
    uint8_t     u8_slave_id;            /* スレーブ ID */
    uint16_t    u16_read_start_addr;    /* リードするレジスタの開始アドレス */
    uint16_t    u16_num_to_read;        /* リードするレジスタの個数 */
    uint16_t    u16_write_start_addr;   /* ライトするレジスタの開始アドレス */
    uint16_t    u16_num_to_write;       /* ライトするレジスタの個数 */
    uint8_t     u8_write_num_of_bytes;  /* ライトするデータのバイト数 */
    uint16_t    aru16_data[MAX_REG_DATA];   /* ライトするデータ */
}req_read_write_multiple_reg_t, *p_req_read_write_multiple_reg_t;
```

・Read/Write multiple registers 応答テーブル(resp_read_write_multiple_reg_t)

```
typedef struct _resp_read_write_multiple_reg{
    uint16_t    u16_transaction_id;     /* トランザクション ID 指定 */
    uint16_t    u16_protocol_id;        /* プロトコル ID 指定 */
    uint8_t     u8_slave_id;            /* スレーブ ID */
    uint8_t     u8_exception_code;      /* 要求に対してエラーを検出した場合に 0 以外を設定。
                                           正常終了の場合は 0 を設定。0 以外が設定された場合、
                                           aru16_read_data は無効となります。 */
    uint16_t    u8_num_of_bytes;        /* 更新したデータのバイト数 */
    uint16_t    aru16_read_data[MAX_REG_DATA];  /* リードデータ */
}resp_read_write_multiple_reg_t, *p_resp_read_write_multiple_reg_t;
```

B. API 仕様書

本章では、本書内で使用されている Modbus スタック関連のアプリケーションプログラミングインターフェース(API)の仕様の詳細について説明します。

1) スタック初期化

Modbus スタックの初期化で使用する API について、以下に示します。

Modbus スタックを Modbus TCP として初期化します。

r_modbus_tcp_init_stack	Modbus TCP スタックの初期化

【書式】

```
uint32_t r_modbus_tcp_init_stack(uint8_t  u8_stack_mode,
                                 uint8_t  u8_tcp_multiple_client,
                                 uint32_t u32_additional_port)
```

【引数】

uint8_t	u8_stack_mode	スタックモード指定
uint8_t	u8_tcp_multiple_client	マルチクライアント対応指定
uint32_t	u32_additional_port	追加ポート番号

【戻り値】

uint32_t	エラーコード

【エラーコード】

ERR_OK	正常終了
ERR_STACK_INIT	初期化失敗

【解説】

本 API では、以下のパラメータを指定します。

・**u8_stack_mode**：スタックの種別を指定します。**MODBUS_TCP_SERVER_MODE** 固定

・**u8_tcp_multiple_client**：複数クライアントからの通信を受け付けるかどうかを指定
指定には以下のマクロを使用します。

マルチクライアント対応指定	意味
ENABLE_MULTIPLE_CLIENT_CONNECTION	マルチクライアント接続有効
DISABLE_MULTIPLE_CLIENT_CONNECTION	マルチクライアント接続無効

・**u32_additional_port**：通信ポートにデフォルト（502）以外を使用する場合に指定
追加しない場合は 0 を指定してください。

Modbus スタックを Modbus TCP-シリアルゲートウェイとして初期化します。

r_modbus_tcp_init_gateway_stack	ModbusTCP-シリアルゲートウェイスタックの初期化

【書式】

```
uint32_t r_modbus_tcp_init_gateway_stack(uint8_t  u8_stack_mode,
                                         uint8_t  u8_tcp_multiple_client,
                                         uint32_t u32_additional_port);
```

【引数】

uint8_t	u8_stack_mode	スタックモード指定
uint8_t	u8_tcp_multiple_client	マルチクライアント対応指定
uint32_t	u32_additional_port	追加ポート番号

【戻り値】

uint32_t	エラーコード

【エラーコード】

ERR_OK	正常終了
ERR_INVALID_STACK_MODE	スタックモードの指定が無効
ERR_INVALID_SLAVE_ID	スレーブ ID が無効
ERR_INVALID_STACK_INIT_PARAMS	ユーザ指定の情報に無効なパラメータがある
ERR_STACK_INIT	スタックの起動に失敗

【解説】

本 API では、以下のパラメータを指定します。

・u8_stack_modeはスタックの種別を指定します。ゲートウェイモードで使用する場合、シリアルデバイスとの通信で使用するモードを指定します。

スタックモード指定	意味
MODBUS_RTU_MASTER_MODE	Modbus Stack RTU マスタモードを選択
MODBUS_ASCII_MASTER_MODE	Modbus Stack ASCII マスタモードを選択

・u8_tcp_multiple_client は、複数クライアントからの通信を受け付けるかどうかを指定します。指定には以下のマクロを使用します。

マルチクライアント対応指定	意味
ENABLE_MULTIPLE_CLIENT_CONNECTION	マルチクライアント接続有効
DISABLE_MULTIPLE_CLIENT_CONNECTION	マルチクライアント接続無効

・u32_additional_port は、通信ポートにデフォルト（502）以外を使用する場合に指定します。追加しない場合は 0 を指定してください。

クライアントからの各ファンクションコードに対する処理要求をユーザ定義の関数に関連付けます。

r_modbus_slave_map_init	Modbus ファンクションコードマッピング

【書式】

uint32_t r_modbus_slave_map_init(p_slave_map_init_t pt_slave_func_req);

【引数】

p_slave_map_init_t	pt_slave_func_req	ファンクションコードマッピングテーブルへのポインタ

【戻り値】

uint32_t	エラーコード

【エラーコード】

ERR_OK	正常終了
ERR_INVALID_STACK_INIT_PARAMS	引数が NULL

【解説】

Modbus スレーブスタックが要求を受信した際、登録された関数を呼び出します。

本 API はスレーブモードの場合のみ有効となります。

・ファンクションコードマッピングテーブル (*_slave_map_init_t*)

```
typedef struct _slave_map_init{
    fp_function_code1_t     fp_function_code1;     /* function code 1 (Read Coils) */
    fp_function_code2_t     fp_function_code2;     /* function code 2 (Read Discrete Inputs) */
    fp_function_code3_t     fp_function_code3;     /* function code 3 (Read Holding Registers) */
    fp_function_code4_t     fp_function_code4;     /* function code 4 (Read Input Registers) */
    fp_function_code5_t     fp_function_code5;     /* function code 5 (Write Single Coil) */
    fp_function_code6_t     fp_function_code6;     /* function code 6 (Write Single Register) */
    fp_function_code15_t    fp_function_code15;    /* function code 15 (Write Multiple Coils) */
    fp_function_code16_t    fp_function_code16;    /* function code 16 (Write Multiple Registers) */
    fp_function_code23_t    fp_function_code23;    /* function code 23 (Read/Write Multiple Registers) */
}slave_map_init_t, *p_slave_map_init_t;
```

各ファンクションコードに対応したコールバック関数は、以下の書式で定義されます。各コールバック関数の引数に使用されている構造体の詳細については Appendix. A を参照してください。

function code 1 (Read Coils)

fp_function_code1_t	Modbus ファンクションコード 1(Read coils)に対応したコールバック関数

【書式】

```
uint32_t (*fp_function_code1_t)(p_req_read_coils_t pt_req_read_coils,
                                p_resp_read_coils_t pt_resp_read_coils );
```

【引数】

p_req_read_coils_t	pt_req_read_coils	Read coil 要求情報が格納された構造体へのポインタ
p_resp_read_coils_t	pt_resp_read_coils	Read coil 応答データを格納する構造体へのポインタ

【戻り値】

uint32_t	0 : 正常,1 : 異常

function code 2 (Read Discrete Inputs)

fp_function_code2_t	Modbus ファンクションコード 2(Read discrete inputs)に対応したコールバック関数

【書式】

```
uint32_t (*fp_function_code2_t)(p_req_read_inputs_t pt_req_read_inputs,
                                p_resp_read_inputs_t pt_resp_read_inputs );
```

【引数】

p_req_read_inputs_t	pt_req_read_inputs	Read discrete inputs 要求情報が格納された構造体へのポインタ
p_resp_read_inputs_t	pt_resp_read_inputs	Read discrete inputs 応答データを格納する構造体へのポインタ

【戻り値】

uint32_t	0 : 正常 ,1 : 異常

function code 3 (Read Holding Registers)

fp_function_code3_t	Modbus ファンクションコード 3(Read holding register)に対応したコールバック関数

【書式】

```
uint32_t (*fp_function_code3_t)(p_req_read_holding_reg_t pt_req_read_holding_reg,
                                p_resp_read_holding_reg_t pt_resp_read_holding_reg);
```

【引数】

p_req_read_holding_reg_t	pt_req_read_holding_reg	Read holding register 要求情報が格納された構造体へのポインタ
p_resp_read_holding_reg_t	pt_resp_read_holding_reg	Read holding register 応答データを格納する構造体へのポインタ

【戻り値】

uint32_t	0 : 正常 ,1 : 異常

function code 4 (Read Input Registers)

fp_function_code4_t	Modbus ファンクションコード 4(Read input register)に対応したコールバック関数

【書式】

uint32_t (*fp_function_code4_t)(p_req_read_input_reg_t pt_req_read_input_reg,
p_resp_read_input_reg_t pt_resp_read_input_reg);

【引数】

p_req_read_input_reg_t	pt_req_read_input_reg	Read Input register 要求情報が格納された構造体へのポインタ
p_resp_read_input_reg_t	pt_resp_read_input_reg	Read holding register 応答データを格納する構造体へのポインタ

【戻り値】

uint32_t	0 : 正常 ,1 : 異常

function code 5 (Write Single Coil)

fp_function_code5_t	Modbus ファンクションコード 5(Write single coil)に対応したコールバック関数

【書式】

uint32_t (*fp_function_code5_t)(p_req_write_single_coil_t pt_req_write_single_coil,
p_resp_write_single_coil_t pt_resp_write_single_coil);

【引数】

p_req_write_single_coil_t	pt_req_write_single_coil	Write single coil 要求情報が格納された構造体へのポインタ
p_resp_write_single_coil_t	pt_resp_write_single_coil	Write single coil 応答データを格納する構造体へのポインタ

【戻り値】

uint32_t	0 : 正常 ,1 : 異常

function code 6 (Write Single Register)

fp_function_code6_t	Modbus ファンクションコード 6(Write single register)に対応したコールバック関数

【書式】

uint32_t (*fp_function_code6_t)(p_req_write_single_reg_t pt_req_write_single_reg,
p_resp_write_single_reg_t pt_resp_write_single_reg);

【引数】

p_req_write_single_reg_t	pt_req_write_single_reg	Write single register 要求情報が格納された構造体へのポインタ
p_resp_write_single_reg_t	pt_resp_write_single_reg	Write single register 応答データを格納する構造体へのポインタ

【戻り値】

uint32_t	0 : 正常 ,1 : 異常

function code 15 (Write Multiple Coils)

fp_function_code15_t	Modbus ファンクションコード 15(Write multiple coils)に対応したコールバック関数

【書式】

```
uint32_t (*fp_function_code15_t) (p_req_write_multiple_coils_t pt_req_write_multiple_coils,
                                  p_resp_write_multiple_coils_t pt_resp_write_multiple_coils);
```

【引数】

p_req_write_multiple_coils_t	pt_req_write_multiple_coils	Write multiple coils 要求情報が格納された構造体へのポインタ
p_resp_write_multiple_coils_t	pt_resp_write_multiple_coils	Write multiple coils 応答データを格納する構造体へのポインタ

【戻り値】

uint32_t	0 : 正常 ,1 : 異常

function code 16 (Write Multiple Registers)

fp_function_code16_t	Modbus ファンクションコード 16(Write multiple registers)に対応したコールバック関数

【書式】

```
uint32_t (*fp_function_code16_t) (p_req_write_multiple_reg_t pt_req_write_multiple_reg,
                                  p_resp_write_multiple_reg_t pt_resp_write_multiple_reg);
```

【引数】

p_req_write_multiple_reg_t	pt_req_write_multiple_reg	Write multiple registers 要求情報が格納された構造体へのポインタ
p_resp_write_multiple_reg_t	pt_resp_write_multiple_reg	Write multiple registers 応答データを格納する構造体へのポインタ

【戻り値】

uint32_t	0 : 正常 ,1 : 異常

function code 23 (Read/Write Multiple Registers)

fp_function_code23_t	Modbus ファンクションコード 23(Read/Write multiple registers)に対応したコールバック関数

【書式】

```
uint32_t (*fp_function_code23_t) (p_req_read_write_multiple_reg_t pt_req_read_write_multiple_reg,
                                  p_resp_read_write_multiple_reg_t pt_resp_read_write_multiple_reg);
```

【引数】

p_req_read_write_multiple_reg_t	pt_req_read_write_multiple_reg	Read/Write multiple registers 要求情報が格納された構造体へのポインタ
p_resp_read_write_multiple_reg_t	pt_resp_read_write_multiple_reg	Read/Write multiple registers 応答データを格納する構造体へのポインタ

【戻り値】

uint32_t	0 : 正常 ,1 : 異常

2) IP アドレスリスト機能

IP アドレスのリスト機能で使用する API について、以下に示します。

IP アドレスリスト機能の有効/無効およびリストに登録された IP アドレスでのアクセス権を指定します。

r_modbus_tcp_init_ip_table	IP アドレスリスト機能の状態設定

【書式】

```
void_t r_modbus_tcp_init_ip_table(ENABLE_FLAG e_flag,
                                  TABLE_MODE e_mode);
```

【引数】

ENABLE_FLAG	e_flag	IP アドレスリストの有効/無効指定 有効: ENABLE, 無効: DISABLE
TABLE_MODE	e_mode	リストに含まれている IP アドレスのアクセス権指定 アクセス許可: ACCEPT, アクセス拒否: REJECT

【戻り値】

void_t

【エラーコード】

—

【解説】

リスト機能を有効としモード指定をアクセス許可とすると、リスト機能はホワイトリストとして動作します。また、リスト機能を有効としモード指定をアクセス禁止とすると、リスト機能はブラックリストとして動作します。

デフォルトでは、IP アドレスリスト機能は無効状態となります。

IP アドレスリストに指定した IP アドレスを追加します。

r_modbus_tcp_add_ip_addr	IP アドレスリストへの追加

【書式】

```
uint32_t r_modbus_tcp_add_ip_addr(uint32_t u32_host_ip);
```

【引数】

uint32_t	u32_host_ip	IP アドレス(IPv4 表記)を指定。 例. GOAL_NET_IPV4(192,168,1,100)

【戻り値】

uint32_t	エラーコード

【エラーコード】

ERR_OK	正常終了
ERR_IP_ALREADY_PRESENT	指定アドレスが登録済み
ERR_MAX_CLIENT	登録数が最大値に達している
ERR_TABLE_DISABLED	ホスト IP リストが無効状態

【解説】

IP アドレスの指定には、GOAL_NET_IPV4 マクロを使用してください。

指定した IP アドレスが IP アドレスリスト機能の対象 IP アドレスかどうかをチェックします。

r_modbus_chk_connectable_ip	IP アドレスのリスト機能判定

【書式】

GOAL_STATUS_T r_modbus_chk_connectable_ip(uint32_t ipaddr);

【引数】

uint32_t	ipaddr	IP アドレス(IPv4 表記) 例. GOAL_NET_IPV4(192,168,1,100)

【戻り値】

GOAL_STATUS_T	エラーコード

【エラーコード】

GOAL_OK	対象 IP アドレスはアクセス許可対象
GOAL_ERR_BUSY	対象 IP アドレスはアクセス拒否対象

3)　TCP 接続管理

TModbus スタックでは、TCP 接続管理を uGOAL が生成する GOAL_ENT_CHAN_T インスタンスで行います。

動作中の Modbus スタックが新たな TCP 接続を受け付け可能か判定します。

r_modbus_tcp_multi_connection	TCP 接続可否判定

【書式】

GOAL_STATUS_T r_modbus_tcp_multi_connection(void);

【引数】		
void	—	—

【戻り値】	
GOAL_STATUS_T	エラーコード

【エラーコード】	
GOAL_OK	接続許可状態
GOAL_ERR_BUSY	接続禁止状態

【解説】

接続可能な状態だと GOAL_OK を返します。マルチコネクション非対応かつ既に TCP 接続済みだと GOAL_ERR_BUSY を返します。

指定した TCP 接続をコネクションリストに登録します。

r_modbus_tcp_reg_connection_list	コネクションリストへの登録

【書式】

void r_modbus_tcp_reg_connection_list(GOAL_NET_CHAN_T *pChan);

【引数】		
GOAL_NET_CHAN_T*	pChan	TCP 接続のインスタンス

【戻り値】	
GOAL_STATUS_T	エラーコード

【エラーコード】	
GOAL_OK	正常終了
GOAL_ERR_FULL	登録済の IP アドレスが上限

【解説】

登録に成功すると GOAL_OK を返します。コネクションリストに登録された TCP 接続が、既に MAXIMUM_NUMBER_OF_CLIENTS で指定された値に達している場合は、登録せず GOAL_ERR_FULL を返します。

指定した TCP 接続をコネクションリストから削除します。

r_modbus_tcp_del_connection_list	コネクションリストからの削除

【書式】

void r_modbus_tcp_del_connection_list(GOAL_NET_CHAN_T *pChan);

【引数】

GOAL_NET_CHAN_T*	pChan	TCP 接続のインスタンス

【戻り値】

void	エラーコード

【エラーコード】

—

ネクションリストに最も古く登録された TCP 接続のインスタンスを返します。

r_modbus_tcp_get_oldest_connection	最も古い TCP 接続の取得

【書式】

GOAL_NET_CHAN_T *r_modbus_tcp_get_oldest_connection(void);

【引数】

void	—	—

【戻り値】

GOAL_NET_CHAN_T*	pChan	TCP 接続のインスタンス

【エラーコード】

—

4) Modbus パケット解析

Modbus パケットを解析する API について、以下に示します。

appl_parseModbusPacket		Modbus パケット解析

【書式】

```
GOAL_STATUS_T appl_parseModbusPacket(
  GOAL_MA_CHAN_TCP_T *pMaTcpHdl,
  GOAL_NET_CHAN_T *pChan,
  GOAL_BUFFER_T *pBuf);
```

【引数】		
GOAL_MA_CHAN_TCP_T *	pMaTcpHdl	MA ハンドラへのポインタ
GOAL_NET_CHAN_T *	pChan	TCP 接続情報が格納されたインスタンスへのポインタ
GOAL_BUFFER_T *	pBuf	TCP パケットが格納された GOAL バッファ
【戻り値】		
GOAL_STATUS_T		エラーコード
【エラーコード】		
GOAL_OK		正常終了

【解説】

本 API は、uGOAL で受信した TCP パケットを Modbus スタックへと引き渡し実行させる、ブリッジ関数的な位置付けの API となります。uGOAL で生成された TCP 情報を元にパケット解析を行いますので、uGOAL で登録された TCP コールバック処理（本サンプルでは tcpCallback が該当）から、データ処理の一部として呼び出します。

スタックが Modbus TCP として初期化されている場合は、パケット解析したのち、初期化時に登録されたユーザ関数がスレーブ動作として実行されます。その後、応答パケットの生成および送信が行われます。

スタックが Modbus ゲートウェイとして初期化されている場合は、受信したパケットを Modbus RTU または Modbus ASCII パケットに変換して、指定されたシリアルスレーブデバイスに送信します。また、スレーブデバイスからの応答パケットを受信すると、受信したパケットを Modbus TCP パケットに変換して、Modbus マスター側に送信します。

C. サンプルソフトウェア起動方法

統合開発環境 e2studio 操作方法を説明します。
事前に 4.評価用ツールを用いた通信テスト を参考に動作させるスタックモードに合わせてハードウェア接続を完了させてください。

本書の説明では、R-IN32M3 モジュール評価環境 SEMB1320 が対象となります。Modbus TCP ゲートウェイモード時に Modbus RTU/ASCII スレーブデバイス例として利用している RX72M 通信ボードについては、『RX72M Group 通信ボード Modbus スタートアップマニュアル(R01AN4862****)』を参照ください。

1) e2studio を起動後、「ファイル」→「インポート」をクリックします。
2) 「選択」ダイアログで「一般」→「既存プロジェクトをワークスペースへ」を選択し「次へ」をクリックします。

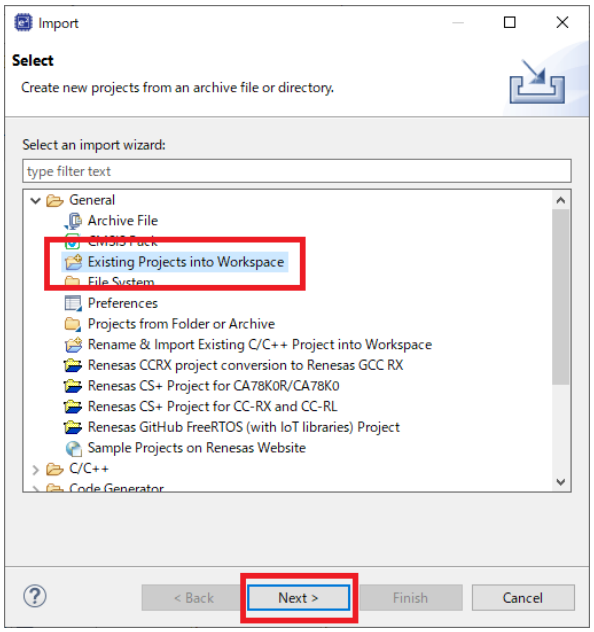


図 C-1　プロジェクトのインポート

3) 「プロジェクトのインポート」ダイアログの「ルートディレクトリの選択」チェックボックスを選択し、「参照」をクリックします。

環境にあったプロジェクトを選択し「開く」をクリック。「終了」をクリックしプロジェクトのインポートを完了します。

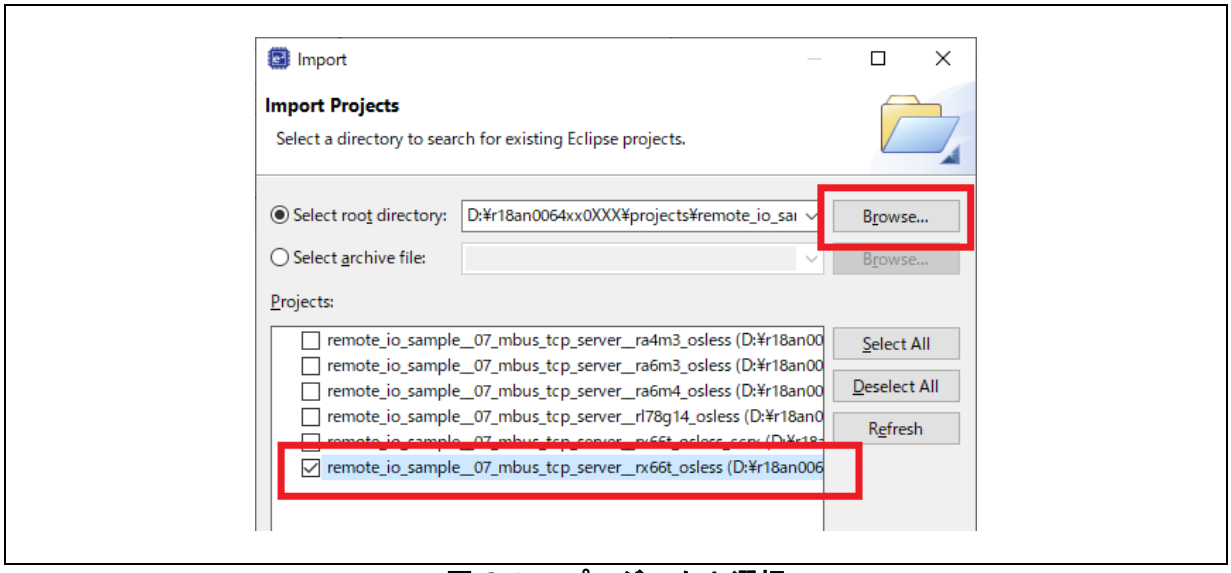


図 C-2　プロジェクト選択

Modbus TCP 向けの プロジェクトファイルは下記ディレクトリに格納されています。

Modbus TCP project File:

RX66T_uCCM_V*** \ projects \ remote_io_sample \ **07_mbus_tcp_server**

4) サンプルプロジェクトを右クリックし、「ビルド構成」の「アクティブにする」から実行するスタックモードを選択します。

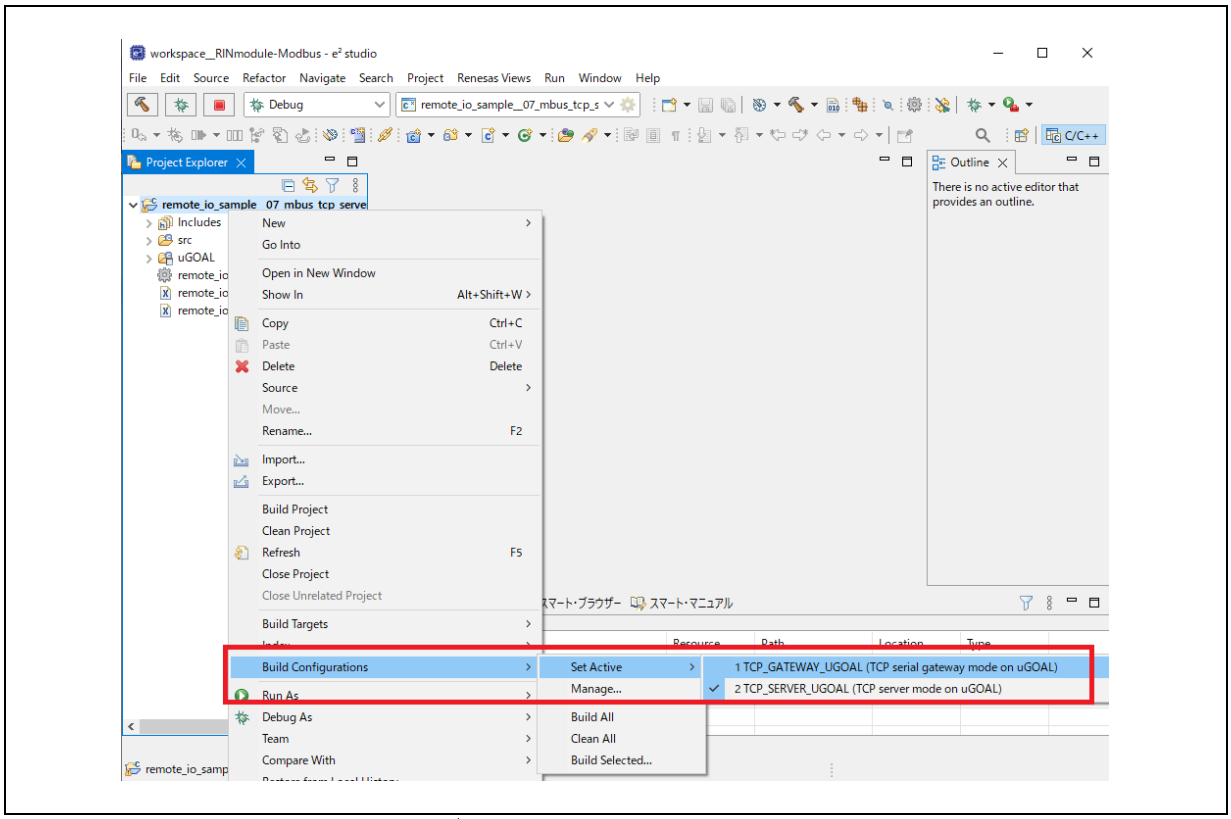


図 C-3　ビルド構成設定 [スタックモード選択]

表 C-1　ビルド構成とスタックモード

ビルド構成名	スタックモード	接続構成
TCP_GATEWAY_UGOAL	Modbus TCP ゲートウェイ スタック	2.6.2 Modbus TCP ゲートウェイ
TCP_SERVER_UGOAL	Modbus TCP サーバスタック	2.6.1 Modbus TCP サーバスタック

5) コンフィグファイルをダブルクリックし、「コンフィグレーション」ウィンドウを開きます。「コードの生成」をクリックし、あらかじめ登録されているドライバを生成させます。「コンポーネント」に登録されている各種ドライバが活性化されている (アイコンが水色となる) ことを確認します。

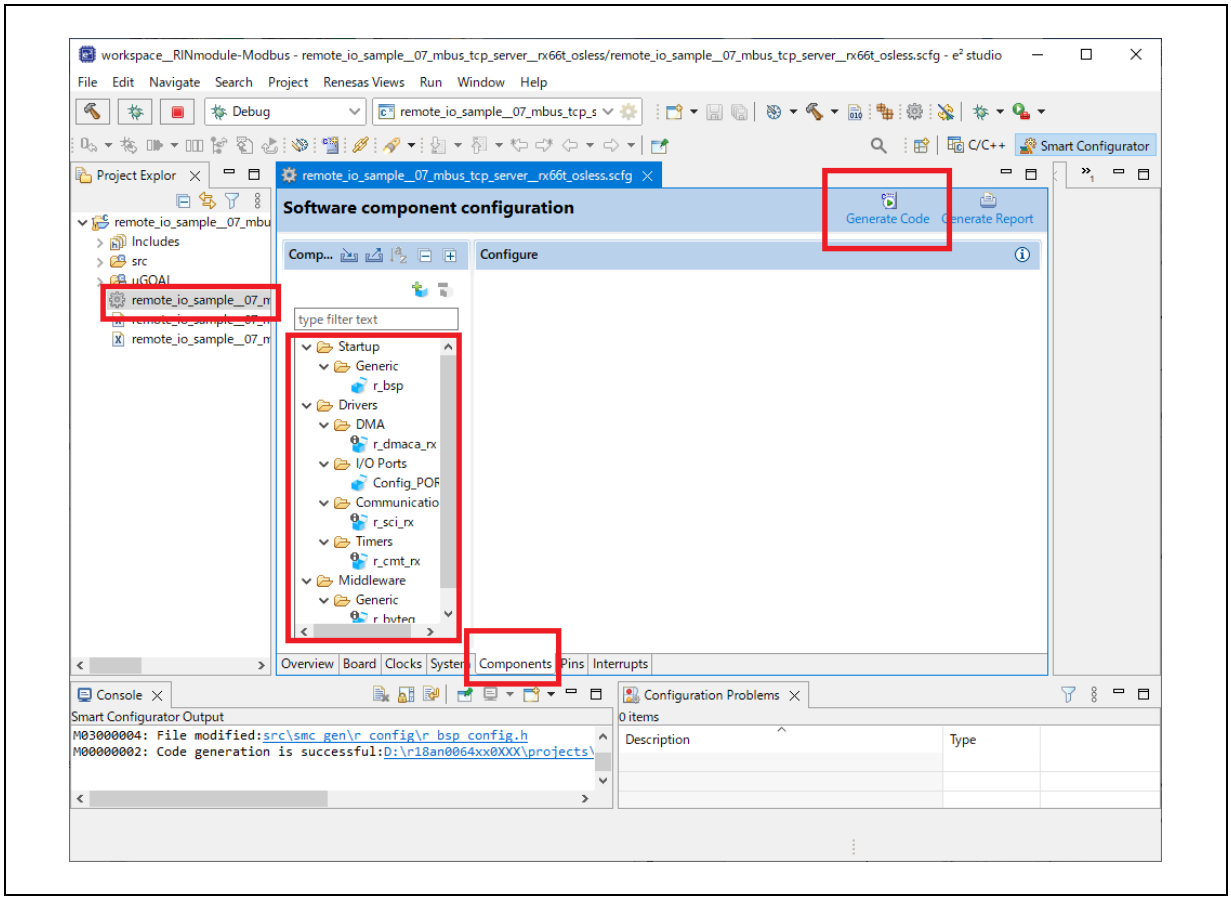


図 C-4　コード生成

6) 「プロジェクトのビルド」を選択し、ビルドを実行します。
エラー無くビルドが終了すれば成功です。

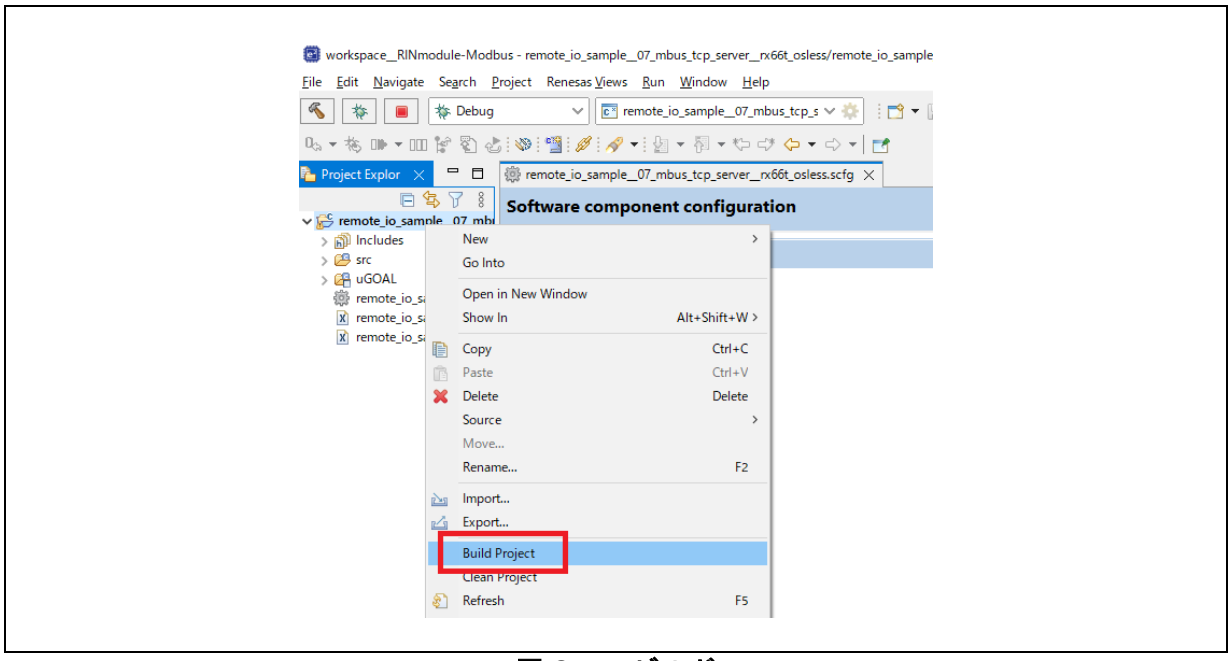


図 C-5　ビルド

7) 「デバッグ」から「Renesas GDB Hardware Debugging」を選択し、プログラムダウンロードを実行します。
エラー無くダウンロードが終了すれば成功です。

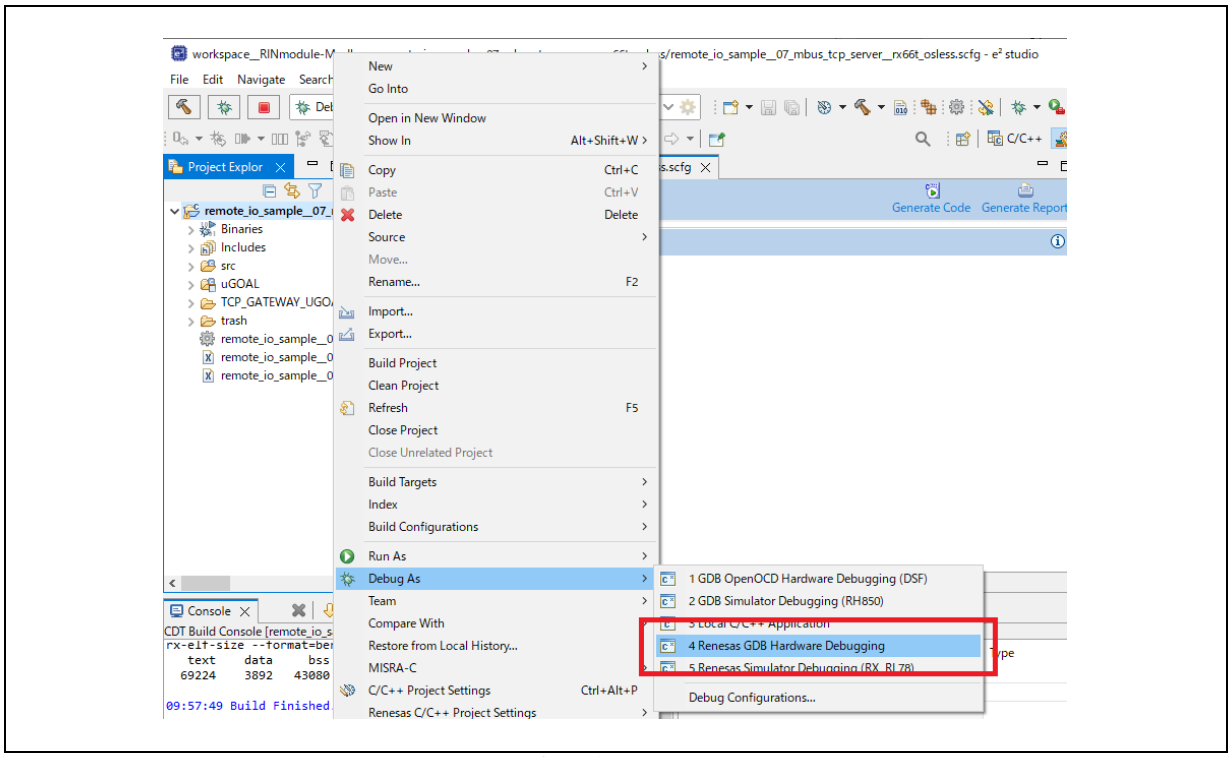


図 C-6　プログラムダウンロード

8) 「再開」をクリックし、プログラムを実行します。

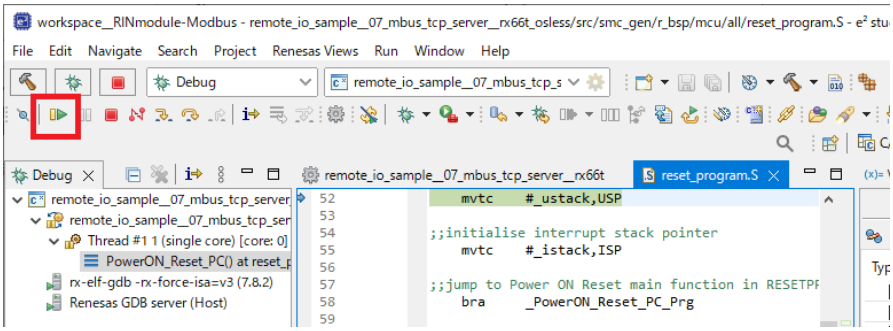


図 C-7 プログラム実行

以上で R-IN32M3 モジュール評価環境 SEMB1320 のサンプルプログラム起動は終了です。

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2022.1.7	-	新規作成
2.00	2022.8.5	-	Modbus TCP ゲートウェイ モード機能対応による説明追記

商標

* Arm および Cortex は、Arm Limited（またはその子会社）の EU またはその他の国における登録商標です。
* Ethernet およびイーサネットは、 富士ゼロックス株式会社の登録商標です。
* EtherCAT は、ドイツ Beckhoff Automation GmbH によりライセンスされた特許取得済み技術であり登録商標です。
* Ethernet/IP は、ODVA, Inc.の商標です。
* PROFINET は、PROFIBUS Nutzerorganisation e.V. (PNO) の登録商標です。
* Modbus は、Schneider Electric SA の登録商標です。
* その他、本資料中の製品名やサービス名は全てそれぞれの所有者に属する商標または登録商標です。

製品ご使用上の注意事項

ここでは、マイコン製品全体に適用する「使用上の注意事項」について説明します。個別の使用上の注意事項については、本ドキュメントおよびテクニカルアップデートを参照してください。

1. 静電気対策
   CMOS 製品の取り扱いの際は静電気防止を心がけてください。CMOS 製品は強い静電気によってゲート絶縁破壊を生じることがあります。運搬や保存の際には、当社が出荷梱包に使用している導電性のトレーやマガジンケース、導電性の緩衝材、金属ケースなどを利用し、組み立て工程にはアースを施してください。プラスチック板上に放置したり、端子を触ったりしないでください。また、CMOS 製品を実装したボードについても同様の扱いをしてください。
2. 電源投入時の処置
   電源投入時は、製品の状態は不定です。電源投入時には、LSI の内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。
3. 電源オフ時における入力信号
   当該製品の電源がオフ状態のときに、入力信号や入出力プルアップ電源を入れないでください。入力信号や入出力プルアップ電源からの電流注入により、誤動作を引き起こしたり、異常電流が流れ内部素子を劣化させたりする場合があります。資料中に「電源オフ時における入力信号」についての記載のある製品は、その内容を守ってください。
4. 未使用端子の処理
   未使用端子は、「未使用端子の処理」に従って処理してください。CMOS 製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI 周辺のノイズが印加され、LSI 内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。
5. クロックについて
   リセット時は、クロックが安定した後、リセットを解除してください。プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。
6. 入力端子の印加波形
   入力ノイズや反射波による波形歪みは誤動作の原因になりますので注意してください。CMOS 製品の入力がノイズなどに起因して、V_{IL}（Max.）から V_{IH}（Min.）までの領域にとどまるような場合は、誤動作を引き起こす恐れがあります。入力レベルが固定の場合はもちろん、V_{IL}（Max.）から V_{IH}（Min.）までの領域を通過する遷移期間中にチャタリングノイズなどが入らないように使用してください。
7. リザーブアドレス（予約領域）のアクセス禁止
   リザーブアドレス（予約領域）のアクセスを禁止します。アドレス領域には、将来の拡張機能用に割り付けられている リザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。
8. 製品間の相違について
   型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。同じグループのマイコンでも型名が違うと、フラッシュメモリ、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ幅射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

© 2022 Renesas Electronics Corporation. All rights reserved.

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器・システムの設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因して生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品、本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング等により生じた損害に関し、当社は、一切その責任を負いません。
5. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。
   標準水準：　コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット等
   高品質水準：輸送機器（自動車、電車、船舶等）、交通制御（信号）、大規模通信機器、金融端末基幹システム、各種安全制御装置等
   当社製品は、データシート等により高信頼性、Harsh environment 向け製品と定義しているものを除き、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。
6. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合および事故につきましては、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は、データシート等において高信頼性、Harsh environment 向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
8. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。かかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
9. 当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
10. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものといたします。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。
12. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

注 1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社が直接的、間接的に支配する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

(Rev.4.0-1 2017.11)

本社所在地

〒135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

www.renesas.com

お問合せ窓口

弊社の製品や技術、ドキュメントの最新情報、最寄の営業お問合せ窓口に関する情報などは、弊社ウェブサイトをご覧ください。

www.renesas.com/contact/

商標について

ルネサスおよびルネサスロゴはルネサス エレクトロニクス株式会社の商標です。すべての商標および登録商標は、それぞれの所有者に帰属します。

© 2022 Renesas Electronics Corporation. All rights reserved.